UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Fine-grained characterization of edge workloads

Magnus Kanck

INF-3981 Master's Thesis in Computer Science - June 2023

Abstract

Edge computing is an emerging paradigm within the field of distributed computing, aimed at bringing data processing capabilities closer to the data-generating sources to enable real-time processing and reduce latency. However, the lack of representative data in the literature poses a significant challenge for evaluating the effectiveness of new algorithms and techniques developed for this paradigm.

A part of the process towards alleviating this problem includes creating realistic and relevant workloads for the edge computing community. Research has already been conducted towards this goal, but resulting workload characterizations from these studies have been shown to not give an accurate representation of the workloads. This research gap highlights the need for developing new methodologies that can accurately characterize edge computing workloads.

In this work we propose a novel methodology to characterize edge computing workloads, which leverages hardware performance counters to capture the behavior and characteristics of edge workloads in high detail. We explore the concept of representing workloads in a high-dimensional space, and develop a "proof-of-concept" classification model, that classifies workloads on a continuous "imprecise" data spectrum, to demonstrate the effectiveness and potential of the proposed characterization methodology.

This research contributes to the field of edge computing by identifying and addressing the limitations of existing edge workload characterization techniques, and also opens up further avenues of research with regards to edge computing workload characterization.

Acknowledgements

I would like to thank my main supervisor, Associate Professor Issam Raïs, for giving me the opportunity to contribute to a very exciting field of research. His advice and guidance has proven invaluable not only for conducting research about the various topics touched upon in this thesis, but also on how to structure the work into a coherent story.

Many thanks to my friends and co-students, with whom I have spent many days and evenings (and sometimes nights) working on assignments, studying for exams or playing billiards.

Finally, and most of all, I would like to thank my partner Naima El bani Altuna for her support and many, many, many words of encouragement. Maite zaitut.

Contents

List of Figures

List of Tables

1 Introduction

Over the past few years the proliferation of mobile computing technologies and applications have led to a push towards distributed computing, in contrast to the centralizing trend seen in cloud computing over the last decade [3, 24]. One such distributed computing framework is the novel edge computing paradigm which lies at the edge of the Internet, close to mobile devices and sensors. A main characteristic of the edge computing paradigm is that a significant amount of computing and storage resources are moved off the cloud to be situated closer to data-generating devices [14].

In recent advancements within the edge computing paradigm, there has been a recognized need for developing realistic and relevant edge workloads for the testing and evaluation of new algorithms and techniques [28]. A part of this process includes the characterization of such workloads and while previous work into this has been done [29, 30], results have shown to be overly generalized, thereby lacking the accuracy needed to classify realistic workloads [28].

The objective for this thesis is to create, present and analyze a methodology that generates fine-grained characterizations of edge workloads. This will be accomplished by leveraging low-level hardware performance metrics, made available via the Performance Application Programming Interface (PAPI) [20] to produce highly-detailed characterizations of edge workloads. To understand why we go in this direction, we will first present and evaluate existing workload characterization techniques, and identify their potential shortcomings.

In addition, a "proof-of-concept" fuzzy classification model to classify edge workloads will be developed, presented and analyzed given the obtained fine-grained characterization.

1.1 Contributions

The contributions of this thesis mainly serve to advance the field of edge computing by identifying and addressing the limitations of existing edge workload characterization techniques. The contributions are as follows:

- This thesis will present and evaluate existing techniques used in the literature for workload characterization, with focus on the edge computing paradigm. The purpose of which is to provide a comprehensive overview of the existing workload characterization techniques and their effectiveness at accurate workload characterization.

- Introducing hardware performance counters (HPCs) as a tool to characterize edge computing workloads, with respect to application performance behaviour.

- Developing, presenting and evaluating a novel methodology to produce fine-grained characterizations of edge workloads, which goes beyond existing characterization techniques used in previous work, to provide a more detailed understanding of the behaviour and characteristics of edge workloads.

- A "proof-of-concept" fuzzy classification model for the purpose of classifying edge workloads with the obtained fine-grained characterization is presented and analyzed. This will serve as a practical demonstration of the effectiveness and potential of the proposed characterization methodology.

The proposed methodology for fine-grained characterization of edge workloads involves several steps. First, we select a set of N hardware performance events that we consider interesting for workload characterization. Using a hardware event performance-monitoring API, we instrument the HPCs to monitor the selected hardware events while our benchmark is being executed.

A statistically significant amount of profiling runs R will be performed to more accurately capture the workload profile. Doing this will reduce the effect of noise appearing as the result of potential variability in the performance behaviour of the workload being profiled.

To eliminate potential noise from thread-migration, the workload is scheduled to run on all available processor units concurrently. Therefore each individual profiling run will output U profiling results, where U is the total number of processors on the system.

Afterwards, we collect the results from each profiling run into a aggregated data set consisting of $M = R * U$ data points with N performance counter values each. The resultant output data represents the profiled workload as as a set of M data points in an N-dimensional performance counter event vector space.

Finally, we perform a rudimentary dimensionality reduction to identify and remove potential dimensions what have little to no informational value with regards to workload characterization.

1.2 Outline

The remainder of this thesis will be structured as follows:

A overview of workload characterization and classification concepts and techniques that currently exist in the literature is given in **Chapter 2**.

Chapter 3 Introduces hardware performance counters, their traditional usage in the high power computing community, and explains their usefulness for the purpose of workload characterization.

We explore the concept of representing workloads as objects in a multi-dimensional metric space in **Chapter 4**, and provide an intuitive example as to why having more quantifiable parameters can prove useful for workload characterization.

Chapter 5 directly builds upon the work presented in Chapter 4 to propose a "proof-of-concept" model to classify edge workloads.

In **Chapter 6** our methodology to produce high-detail workload characterizations is presented, along with a worked example where we go through the steps required to characterize a workload using the proposed methodology.

Chapter 7 will give an overview of the various techniques and design choices made when implementing the proposed characterization methodology in Chapter 6, as well as for the proposed classification model in Chapter 5.

An overview of benchmarks, parameters and the testbed used to perform the experiments is given **Chapter 8**.

Chapter 9 will present and analyze the results from experiments performed.

In **Chapter 10** we discuss various advantages and disadvantages of the proposed fine-grained characterization methodology.

Chapter 11 gives an overview of future avenues of research that build upon the work presented in this thesis.

Finally, **Chapter 12** will conclude the thesis.

2

Related Work

In this section the related work regarding the definition, characterization and classification of workloads will first be presented. Then, similar work on workload characterization for paradigms that lie close to and within the edge computing domain will be presented.

The workload characterization and classification techniques presented in this section will be iterated upon further on in this thesis to eventually define a methodology that can produce highly-detailed characterizations of workloads. The focus will be on the characterization and classification of workloads belonging to the edge computing paradigm.

2.1 Surveys on workload characterization and classification in the literature

Several surveys have been conducted to provide an overview of the state of the art regarding workload characterization and classification. In [26], Shishira et al. define workloads as "a computing task that requires access to a mix of resources including processing power, storage, disk input/output(i/o) and network bandwith". However, they also observe that there are no general definitions of workloads in the literature and that "the job/task granularity determines the definition of workloads".

Calzarossa et al. [7] define "workload characteristics" as the demands placed by a task/job on various system resources, described by a set of parameters which can be used to quantify the workload based on their magnitudes. To capture the behaviour of workloads in detail, the authors suggests that "it might be necessary to insert into the system probes, such as event counters". Minimizing the intrusiveness and overhead caused by the instrumentation system is also of importance when performing such measurements.

Shishira et al. [26] provide an overview of workload taxonomies based on different criteria. Of particular interest is workload classification based on *resource requirement*, where workloads can placed into one, or a combination, of the following categories: "Memory workloads", "CPU workloads" "I/O workloads", and finally "Database workloads".

2.2 Characterization of high-power computing/supercomputer workloads

With regards to workload characterization on supercomputers, a static and dynamic workload characterization study is performed by Pasquale et al. [22]. The authors use over a million trace records to create a static characterization of jobs in terms of CPU time, channel I/O time and memory space-time product using relatively extensive categories ("Pub-Low", "Pub-Medium", etc). The characterization is performed by creating a representative job for each command, with the resource consumption being the average of all jobs sharing the same command name. A dynamic characterization is also presented that identifies characteristic time periods, with regards to workload intensities, over an average weekday. The resulting static characterization is relatively coarse-grained, with only three defining parameters, considering the high-detail characterizations we aim to produce in this thesis.

Li et al. [15] in their work perform workload characterization of a multi-cluster supercomputer dedicated to parallel and distributed computing research. They analyze twelve-month workload traces and include characteristics such as "system utilization", "job arrival rate", job interarrival time" and "job cancellation rate". They also look at characteristics specific to job execution, but only take into account "actual runtime", "memory usage", "number of requested processors", and the correlations between them.

2.3 Characterization of cloud and datacenter workloads

Regarding the characterization of cloud workloads, Calzarossa et al. [8] identify "resource usage", "arrival process" and "workload patterns" as relevant research questions, with the main targets being "jobs/tasks" and "Virtual Machines". This statement is also corroborated upon in [26]. Focusing on characterization of *cloud applications*, Calzarossa et al. [8] mention "resource usage" as a key aspect, and that most studies utilize a so-called *task shape*, a multidimensional representation of resource usage. With respect to this they cite Mishra et al. [19] as an example, where *qualitative coordinates* are used to define the dimensions of the *task shape*, which are later used for *task classification*.

A methodology for classifying datacenter workloads is presented in [25]. The authors use two datasets, *Google Cluster Trace* and *Big Brains Trace*, and implement an algorithm that ranks the trace attributes (task duration, CPU usage, memory usage, and disk usage) in terms of predictive power. They use *k-means clustering* and the *elbow criterion* to find the per-attribute clusters (e.g. "low", medium low, "medium", etc) for each of the traces. Finally, a variety of machine learning algorithms are applied to classify the workloads given the obtained classification model.

2.4 Characterization and classification of edge workloads

Several studies have focused on workload characterization for edge workloads and similar paradigms, such as Toczé et al. [30] where broad categories (e.g. "high"/"low") of more abstract characteristics, such as *computation* and *communication demand*, are used as part of their work to create a methodology for edge workload characterization. McChesney et al. [18] also employ similarly broad characteristics in their work to develop *DeFog*, a benchmarking suite for the fog computing paradigm, which is very closely related to the edge computing paradigm.

Toczé et al. [28] iterates further on their work in [30], to define a set of 4 workload classes based on *communication demand* and *computation demand* intensities ("high"/"low"). Traces of edge applications with similar characteristics are then gathered and analyzed. The authors discover that the descriptions of the selected applications in the literature does not align with how they actually behave.

In [16], Limaye and Adegbija perform a characterization of a selection of Internet of Medial Things (IoMT) applications and algorithms from a edge computing perspective. They utilize hardware performance counters to analyze the workload execution behaviour with respect to *compute*, *memory* and *branch* characteristics. They find that the execution behaviour of the workloads differed significantly from the embedded system benchmark suite MiBench [13], indicating the need for new benchmarks suites for IoMT microarchitecture research.

3

Hardware performance counters

Previous attempts at edge workload characterization have generally opted for broader and low-dimensional characteristics to define the behaviour of edge workloads [30, 18]. Preliminary results have shown that such characterizations do not align with the actual behaviour of edge workloads.

In this thesis we wish to "attack" the problem from the bottom-up, so to speak, and leverage low-level hardware performance counters for the purpose of workload characterization.

3.1 Traditional usage

Hardware performance counters (HPCs) are special registers built into microprocessors that can be instrumented to monitor hardware-related metrics, or *events* [5]. Almost all major processing platforms today have built-in HPCs that can provide information about the behaviour of applications, and are extensively used by developers in the *high-performance computing* community to conduct performance analysis [4]. Examples of such performance analysis includes techniques such as application fine-tuning, optimization, monitoring, benchmarking and performance modeling [5].

3.2 Hardware *events*

Mucci et al. [6] define hardware *events* as "...occurrences of specific signals and states related to a processor's function". Examples of such events include *instruction counts*, *cache misses* and *branch mispredictions*, etc. By instrumenting the HPCs to monitor such events, the user can gain access to the underlying behaviour of an application terms of *compute-intensity*, *cache-* and *table lookaside buffer* performance, *branch performance*, among other interesting aspects.

In terms of monitoring hardware events for the purpose of workload characterization, the intuition is that having a highly detailed description of the performance behaviour of workloads based on measurable properties might expose similarities and/or differences between the workloads that exist in the literature, thereby leading to more accurate workload characterizations in the future.

4

Workload dimensions

In this chapter we present the concept of *task shapes*, and how measurable properties of workloads can be used to define a metric space, called the attribute space. Further, an abstract example is given in which increasing the dimensionality of workloads can prove useful for the characterization of edge workloads. Finally, we explore the problem of measuring distances in high-dimensional spaces, and give descriptions of a few distance measurement methods.

4.1 Representing workloads in a multi-dimensional space

Regarding work on characterization of cloud workloads, some authors [19, 9] use the term *task shape* to refer to a multi-dimensional representation of the resource usage of cloud applications. Mishra et al. [19] in their work define a 3-dimensional task shape in terms of *runtime in seconds*, *CPU usage in cores* and *memory usage in gigabytes*, but note that the task shape can have even higher dimensions if other resources are considered.

In this thesis we will define the dimensions of the task shape in terms of hardware events, and thereby extend the *task shape* to a higher dimension. We will also define the *space* that the task shapes live in as the *attribute space*, the coordinates of which are defined by a set of attributes, which we will henceforth

denote as α_n. Specifically, we define the attribute space as the N-dimensional space R^N, where N is the total number of hardware events used for characterization. We are only interested in the positive orthant of R^N as the magnitude of the hardware performance events, or attributes, can only be positive.

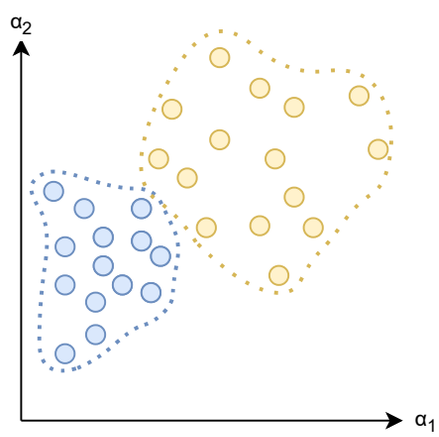

Figure 4.1: Two workloads represented by several data points in an *attribute space* defined by two arbitrary attributes α_1 and α_2. The task shapes are represented by blue and yellow dashed lines respectively.

Our task shapes will also consist of many data points to get a distribution for each workload (Fig. 4.1), which will allow us to capture statistical metrics such as variance and mean along each dimension, and reduce the impact of outlier data points.

Increasing the dimensions of the task shapes can provide a deeper understating of the performance behaviour of workloads. To strengthen this argument, we provide an abstract example in the next section where the the inclusion of an additional attribute to define the task shapes reveals a stark difference between previously similar workloads.

4.2 Increasing dimensions: An example

The task shape of three distinct workloads (red, green and blue circles) are shown on a 2-dimensional subspace of the attribute space R^N (Fig. 4.2). The three task shapes represent *memory-* *cpu-* and *network-intensive* workloads respectively.

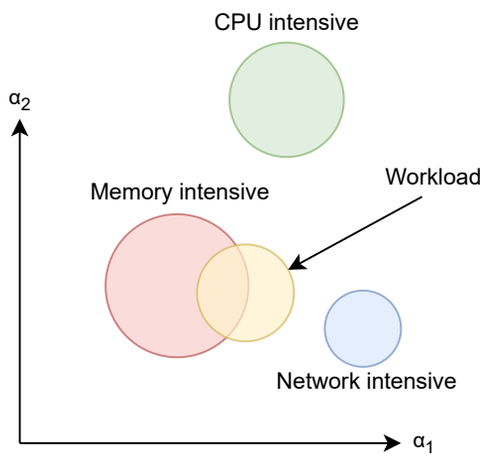

Figure 4.2: Representation of different task shapes in a an *attribute space* defined by two arbitrary attributes α_1 and α_2. The red, green and blue circles are the task shapes of *memory-*, *cpu-* and *network-intensive* workloads respectively. The task shape of the new workload, represented by a yellow circle, must necessarily be be categorized as *memory-intensive* given the apparent similarity with the memory intensive workload.

We introduce a new workload, the task shape of which is represented by a yellow circle in the middle of Fig. 4.2. In this scenario, given the limited information about the behavioural characteristics of the workload, it would be natural to classify the red- and yellow coloured task shapes as belonging to the same category. I.e. the workload represented by the yellow task shape would the categorized as memory intensive.

We increase the dimensionality of the 2-dimensional subspace of R^N given above and introduce a third attribute α_3 (Fig. 4.3). It is now revealed that the task shapes of the new workload and the memory-intensive workload differ in magnitude along the new axis, indicating that they might belong to different workload categories. Increasing the dimensions even further might reveal other similarities or dissimilarities between the task shapes.

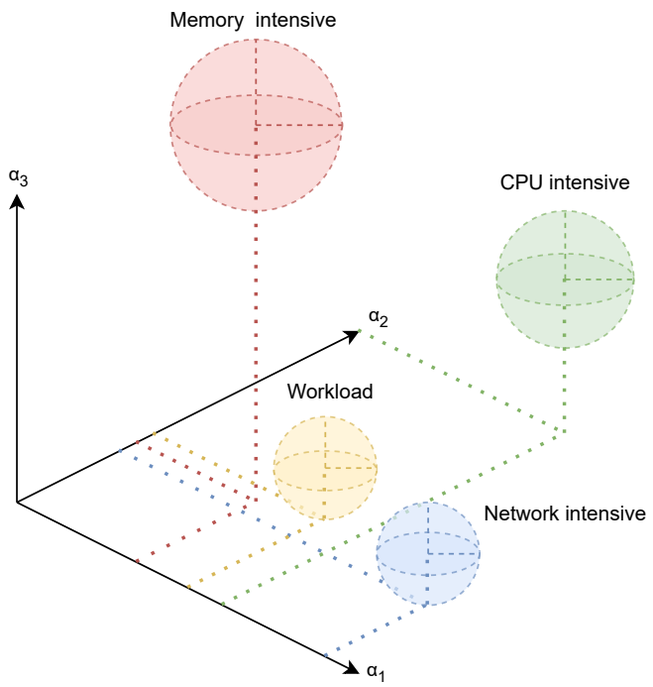

Figure 4.3: Adding a new arbitrary attribute α_3. Increasing the dimensionality of the attribute space might reveal differences and similarities between the various task shapes.

4.3 Measuring similarities in high-dimensional spaces

Workload classes are often defined by few, and sometimes theoretical, properties in the literature. This can lead to situations where the classes are too rigidly, or arbitrarily, defined, thus leading to misclassification of workloads. We propose instead to classify workloads in terms of their closeness to other known task shapes in an N-dimensional *attribute space*, as described in Section 4.1. There are many ways to measure distances in metric spaces, such as the *attribute space* considered in this thesis, and we will consider a few of them here.

4.3.1 Euclidean distance

The most commonly used metric to measure distances between two data points is the *Euclidean distance*, which calculates the length of a line segment between two points p and q. The Euclidean distance function can easily be applied in

our *attribute space* to measure the similarity between two task shapes. The most straightforward way to do this is to define a "center point" for each task shape, and then calculate the Euclidean distance between these points.

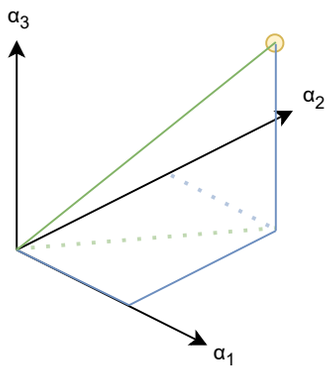

Figure 4.4: Manhattan distance vs. Euclidean distance when measuring the distance from the origin to a point represented by the yellow circle. Here, the Manhattan distance is represented by the three blue lines at right angles to each other, and the Euclidean distance is represented by a straight green line.

4.3.2 Manhattan distance

Also known as *taxicab geometry*, the Manhattan distance metric is defined as the sum of the absolute difference between two points along each dimension. This means that the distance is measured along the axis dimensions at right angles (Fig. 4.4), as opposed to Euclidean distance which measures the distance along a straight line. We consider the Manhattan distance as an alternative to the Euclidean distance in this thesis, as some research [1] indicates that the Manhattan distance may be better suited for distance measurements when dealing with high-dimensional data.

4.3.3 Cosine similarity

The *Cosine similarity* calculates the cosine of the angles between two non-zero vectors in an inner product space, and so gives a measurement of the similarity or dissimilarity between them. The cosine similarity exists in the interval $[-1, 1]$, if the two vectors are proportional then the cosine similarity would be 1, opposite ones would result in a value of -1. Again, we can define a "center point" for each task shape and use that as a vector for calculating the cosine similarity.

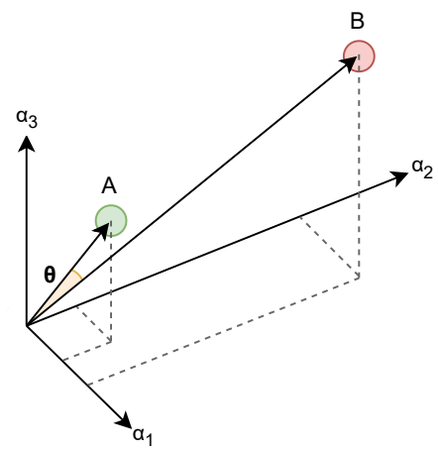

Figure 4.5: Two task shapes (red and green circles) represented in an attribute space defined by three arbitrary attributes α_1, α_2 and α_3. The Cosine similarity uses the angle θ between the two vectors to measure the similarity between the task shapes.

Note that the Cosine similarity does not take into account the magnitude of the vectors, only the angle between them. However, this type of measurement can be useful to find similarities between task shapes have similar ratios between the various attributes, but different magnitudes as shown in Fig. 4.5.

4.4 Defining a center point

In the previous section we assumed that each class task shape had a well-defined "center point" with which various distance measurements, such as the Euclidean distance and Cosine similarity, could be calculated. However, defining such a "center of mass" can also be done in several ways, and we will consider a few of them here.

4.4.1 Geometric median

The Geometric median for a discrete set of points in an Euclidean space, such as our attribute space, is defined as the point which minimizes the sum of distances to each point in the set. The *Geometric median* is the generalization of the median for one-dimensional spaces to higher dimensions. It has been proved there is no explicit formula that can calculate the Geometric median [10]. However, it can be approximated to a relatively good degree [31], but can be somewhat time-consuming.

4.4.2 Greedy median

A faster approach to define the center point of the task shapes involves simply finding the median along each attribute axis, one at a time. This method is greedy in that it only takes into account the optimal choice per dimension, and so the resulting center point will not be globally optimal. I.e. the resulting center point will *not* minimize the sum of distances to each point in the set, and will only be an approximation.

5

Classification of highly detailed workloads

In this chapter a "proof-of-concept" workload classification model is presented. Building upon some of the concepts and techniques outlined in Section 4 to obtain high-dimensional workload characterizations, the first objective of the model is to construct a set of workload classes consisting of workloads that have similar task shapes. These datasets will then serve as representatives for the workload classes. Second, the constructed class representatives can then be used to classify other workloads through the use of a nearest neighbour fuzzy classification scheme.

5.1 Constructing representative workloads

In the context of the *attribute space*, as outlined in Section 4.1, it would be useful to have existing task shapes with known performance behaviours, which can then be used as a "ground truth" for classifying other workloads later on. We argue that the best way to construct these "ground truths" would be to combine several workloads with a similar performance behaviour, or task shape, into a unified class representative, as this will allow us to capture the variance, mean and median along each attribute dimension for each workload class.

If a set of class representative datasets can be constructed, then any future workload can be classified in terms of their closeness to each class representative. In other words, the closer a workload lies to a class representative, the more likely it is to be part of that class.

There are several ways to go about constructing the class representatives, and two such approaches will be presented and evaluated below.

5.1.1 Method 1

Via careful analysis of the descriptions of workloads that exist in the literature, groupings can be made based on edge application use case and/or attribute magnitude. An example of edge workload classification based in part on application use case can be seen in [30]. We also refer to [3] for a comprehensive list of various edge computing use cases.

For our study we make the assumption that applications with the same edge use case will also have similar task shapes. After creating the groups, each individual workload is profiled to create fine-grained characterizations, which are then used to create the class representatives. Depending on the desired class granularity and attribute variance, the classes can be further subdivided, as shown in Fig. 5.1.

The immediate drawback of the this approach is that the definition of the class representatives rests in large part on a descriptive behaviour of the workloads, which might not be fully indicative of their actual performance behaviour.

However, this approach has its merits as well, as the attribute magnitudes of the class representatives will be application use-case specific. I.e. workloads will be classified based on use case, then subdivided based on attribute magnitudes. The advantage of this is that the magnitudes are evaluated (e.g. High, Medium, Low, etc.) relative to the specific application use case.

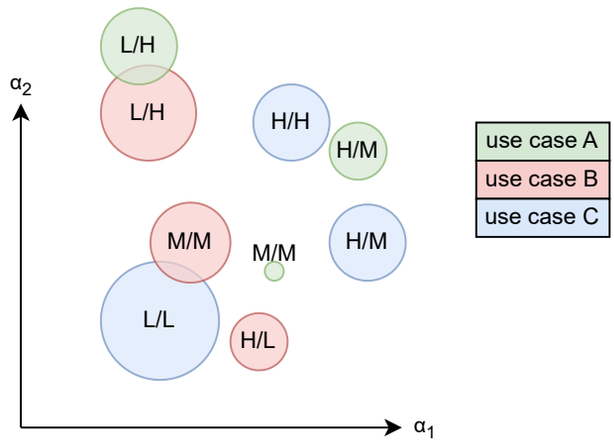

Figure 5.1: Class representatives characterized by two arbitrary attributes α_1 and α_2. The classes are grouped first by use case (A, B or C), then by attribute magnitudes (Low, Medium, High). The magnitudes are evaluated relative to the specific use case when using this approach.

5.1.2 Method 2

Perform the profiling step first, then create the workload groupings based on analysis of the high-detail workload profiles (Fig. 5.2).

This approach creates groupings based on attribute magnitudes only. The advantage of this approach is that the resultant class representatives are defined solely by measurable metrics, and so the configuration can easily be reproduced. However, since applications will exhibit different behaviours depending on their edge use case, the class representatives will possibly end up with having large variances for some attributes, meaning that they might not be able to sufficiently represent outliers that they will inevitably encompass. This can potentially be alleviated by increasing the granularity of workload classes.

Centroid-based clustering models, such as *k-means clustering*, are typically used to achieve such groupings. Using a clustering method has the additional benefit of having a well-defined center point for the class representative, a concept which we introduced in Section 4.4.

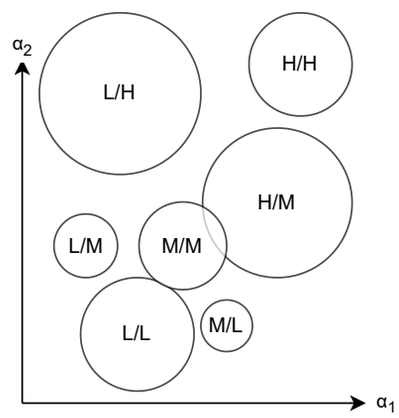

Figure 5.2: Class representatives characterized by two arbitrary attributes α_1 and α_2. The attribute magnitudes (Low, Medium, High) are evaluated irrespective of application use case, potentially leading to "broader" classes.

5.2 Fuzzy workload classification

Applications on the edge generally utilize all components of the system architecture in various ways depending on the use case. An application that is heavily network-intensive, for example, might also exhibit behaviours that are indicative of other workload classes, such as compute-intensive applications.

A classification model that retains information about a workloads similarity to each of the classes can be particularly useful in situations where a task exhibits behaviour that is indicative of two or more class representatives to a relatively equal degree, as shown in Fig. 5.3. In the depicted example we argue that a "one-of" classification model would not be able to accurately classify the given workload due to its nuanced behaviour. Instead what we want to have is a classification model that can correctly classify workloads, but also retain information about a workloads degree of similarity to other classes as well.

In the example classification scenario in Fig. 5.3 our proposed classification model would report that the workload is most similar to the "L/H" class, but also shares many similarities with both the "L/M" and "M/M" classes and thereby indicate that the workload in question exists in a space between the classes. Such information could lead to the creation of new workload classes or a redefinition of existing ones, depending on the desired granularity of classes.

The problem then becomes how to classify edge workloads in a way that not

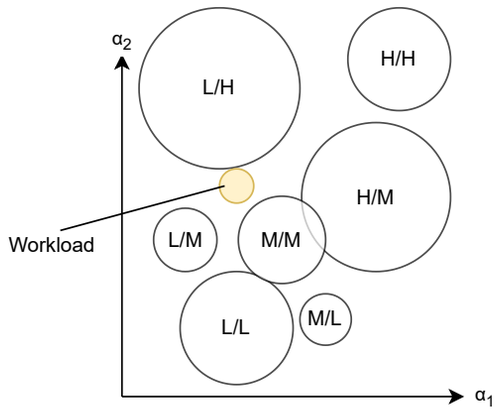

Figure 5.3: Class representatives characterized by two arbitrary attributes α_1 and α_2. "One-of" classification models would classify the workload represented by the yellow task shape as part of the class "L/H".

only identifies the most likely class, but also manages to describe a workloads similarity to other classes. We can borrow the mathematical concept of *fuzzy membership functions* [35], and classify workloads by grading how much they belong to each class on a continuous "truth" scale from 0 to 1. What we mean by this is that we wish to classify workloads in an open, "imprecise" data spectrum, which makes it possible to make several conclusions about the data.

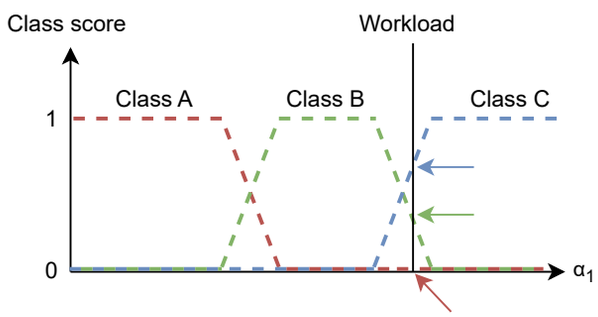

Figure 5.4: Applying the concept of fuzzy membership to classify a workload based on a single attribute α_1. A score of 1 indicates that the workload completely belongs to a class, and a score of 0 indicates the opposite. In this example the workload would have a score of 0.6 towards class C and a score of 0.4 towards class B, while class A would get a score of zero.

We can imagine the classes as existing within a range with a given upper and lower bound along each attribute dimension. In Fig. 5.4, we show a very basic example of this along some arbitrary attribute dimension α_1. Here we see that class A has a high truth value for the lower attribute magnitude spectrum of the dimension α_1. Class C, on the other hand, has a high truth for the upper end of

the spectrum. Finally, class B has a high truth value for attribute magnitudes in between. As we increase the value of α_1 the "degree of truth", or class score, for each class changes, but the total sum of scores is always 1.

Based on the attribute magnitude ranges in which the classes "exist", and the measured attribute value α_1 for the workload, we can calculate "degree of truth" scores based on how close our workload lies to each class.

In reality we won't encounter simple scenarios such as the one depicted in Fig. 5.4. Instead what we might have are more complicated configurations (Fig. 5.5) in which the "degree of truth" of several classes "overlap", in addition to having to deal with many more dimensions.

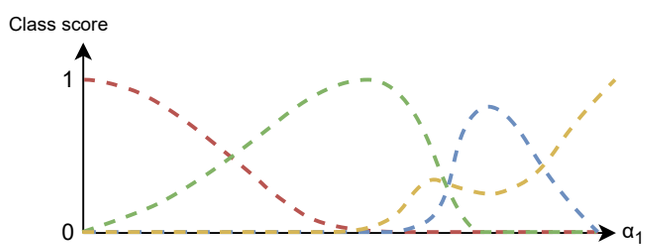

Figure 5.5: A scenario where the attribute magnitude ranges for a set of 4 classes along a single attribute dimension α_1 might be harder to define.

6

Methodology

The main objective of our proposed characterization methodology, presented in this chapter, is to produce high-dimensional task shapes based on measurable properties, which exposes the performance behaviour of any given workload in finer detail. We provide a step-by-step example where, given an arbitrary workload, we characterize it with the proposed characterization methodology.

We extend the worked example to a classification problem later on, where we attempt to group and classify the task shapes of several workloads.

6.1 Fine-grained workload characterization: A worked example

Assuming we have an edge computing task, and we wish to obtain a characterization of it so that we can understand its behaviour and its system hardware requirements. The first step will be to select a set of attributes to define the attribute space in which the task shapes that represent the workloads are defined. Hardware performance counters (HPCs), which are available on most modern systems, serve this purpose well, as they can be instrumented to monitor various metrics (hardware events) related to the performance behaviour of the workloads. There are various ways to access and instrument the HPCs, but it is usually done through the use of an API such as *PAPI* or *perf_events*.

6.1.1 Attribute selection

We select a set of N hardware events that cover a wide variety of system components, such *branch*- and *cache* performance, *instruction counts*, etc. and profile the workload in question. The assumption is that if more and varied events are monitored, the resulting characterization from profiling will more accurately capture the actual performance behaviour of the workload. Typical hardware events to monitor include *last level cache* (LLC) miss/hit rates, *branch instruction* hit/miss rates and derived events such as *instructions per cycle*.

6.1.2 Profiling

We desire to construct a highly detailed characterization of our supposed workload, and therefore opt for a relatively large selection of performance events to monitor. However, monitoring more performance events than the number of available HPCs forces the system to use event counter multiplexing, which introduces a certain loss in precision. Other factors will also inevitably cause noise in the data for various reasons. We therefore choose to profile our workload a statistically significant number of times R, to reduce the impact of noise on the resulting characterization.

It is also desirable to minimize the overhead from thread migration, and we therefore wish to utilize all available processors in order to eliminate this occurrence. Assuming the system on which we profile our supposed workload has a number of CPU cores U, we configure a *job schedule* such that our task will be running concurrently on each core. We can now choose to take the average, or median, profiling result of each core and consider it as a single data point. Or, we can consider the resulting data from each core as individual data points, resulting in U data points per profiling run. We want capture the behaviour of our workload as accurately as possible, and having more data points M greatly contributes towards this goal. Therefore we choose to consider the profiling results from each core as individual data points.

6.1.3 Aggregation

We now have enough information about our assumed edge computing workload to define a *task shape*, which is a multi-dimensional data set that represents our workload in an *attribute space*. In our worked example the attribute space will have N dimensions, i.e. the number of chosen hardware events that describe the behaviour of the workload. Naturally, the task shape will also have N dimensions and will consist of $M = R * U$ data points, where R is the number of profiling runs and U is the number of CPU cores. This task shape is essentially

the fine-grained workload characterization that we are after. We can now, for example, extract useful statistical metrics, such as the mean and variance along each attribute axis, or start comparing our workload to other edge computing workloads.

6.1.4 Analysis

The task shape can also be further refined by identifying and discarding redundant dimensions. That is, attribute, or hardware events, with little to no informational value. If, for example, two of our chosen hardware events are perfectly positively or negatively correlated, then that means that one attribute can be perfectly inferred from the other. We can therefore safely discard one of these attributes without losing any informational value regarding the behaviour of our workload.

6.1.5 An iterative process

At this point the characterization can be considered to be concluded. However, if it is so desired, the steps outlined in Sections 6.1.1 — 6.1.4 can be performed iteratively several times. With each iteration, a new combination of hardware events can be chosen to profile the workload, thereby extending the dimensions of the task shape from the preceding iteration. After concatenating the new and the previous task shape, we can again perform attribute reduction to remove redundant dimensions. And so with each iteration a new combination of hardware events are monitored and analyzed until the desired task shape dimension, or characterization granularity, is achieved.

6.2 Workload classification: Working our example further

We will now assume that the characterization methodology described in the previous section has been performed on a variety of edge computing workloads. The resulting task shapes are defined by the same hardware events, and have roughly the same amount of data points.

Based on the apparent similarities or dissimilarities between these task shapes, we wish to construct well-defined classes of workloads which can be later used to classify future workloads. Depending on the overall objective and desired class granularity, the task shapes can be grouped using a variety of

methods.

6.2.1 Class definition

In this example the various tasks have one of three distinct edge computing use-cases, such as *video data analytics* or *augmented reality*, and so we elect to group them as such. To further increase our class granularity, we might opt to subdivide the classes based on attribute magnitudes via clustering methods such *k-means clustering*, or some other heuristic method.

We now have our well-defined class representatives and can use them to classify future workloads based on similarity measurements. We assume that, for our worked example, the Euclidean distance metric can provide meaningful measurements. If our task shapes consisted of data points with a lot of variance, i.e. the data points are spread out over a large range, or the dimensionality of our task shapes is considerably large, then it might be more beneficial to use other distance metrics.

6.2.2 Classification

We introduce a new workload, also represented by a task shape with the same dimensionality as our class representatives. To classify this new workload, we compute the distance from the center point of the new task shape to each of the class representatives and identify the class representative for which the distance to the new workload is the shortest.

The choice of task shape center point and distance metric used will invariably have an effect on the resulting classification. If computation speed is desired over accuracy, an approximation to the geometric median of the task shapes can be used. For distance measurements, the Euclidean distance can be considered due to being easy to implement and interpret. However, for more larger and more complex data sets, other distance functions should be considered instead.

6.2.3 Fuzzy classification

At this point we could decide that since the new workload lies closest to some class representative, then it belongs to only that class. However, our new workload might also, to a lesser degree, exhibit properties that are more in line with other class representatives. When using a "one-of" classification approach we are unable to capture this property of our data.

Instead what we want to do is to look at these distances relative to each other, and classify our new workload in terms of its degree of closeness to each of the class representatives. We can leverage the inverse of the computed distances to compute a set of weighted scores, one per class, that sum up to one. What we end up with are a set of classification, or "degree of truth", scores for the task shape in question that indicate the degree of similarity to each class.

7

Implementation

We begin this chapter by presenting the various techniques used to implement the characterization methodology in Chapter 6. Further, design choices made during implementation of the classification model is also given here.

7.1 Instrumenting the code

To instrument the HPCs, we made use of the PAPI performance-monitoring library [5, 20]. Specifically, the PapiEx command-line interface extension for PAPI [21].

The exploratory nature of the work presented in this thesis necessitates a naive approach towards the selection of hardware events that will define the attribute space, given the amount of PAPI preset events and architecture-specific native events the library can make available for monitoring. The PapiEx command-line interface provides a starting point by automatically selecting a set of interesting hardware metrics (Table 7.1 and Table 7.2), which will be used to characterize workloads in Section 9.

The hardware events are grouped into preset and native events. The latter are events for which the event codes are platform dependent, while the former are events for which the codes function as symbolic mappings in PAPI to native countable events. We also further subdivide the events according to which com-

ponent of the system architecture they give insight to, such as *branch instruction events*, *level 1 and 2 cache events*, etc.

7.2 Multiplexing

The full set of performance events that PAPI can make available is dependent upon the hardware of the system on which the profiling is done. For our testbed PAPI reports a total of 59 available events. Considering that the aim of this thesis is to produce highly detailed workload characterizations, one might conclude that the optimal choice is to measure all available events when profiling the workload.

However, only a certain amount of events can be monitored simultaneously, dependent on the amount of HPCs offered by the system architecture. By using a technique known as *hardware event multiplexing*, which is automatically supported by PapiEx, the number of monitored events can be increased beyond the number of HPCs. The idea is to allocate the events across the HPCs in a round-robin time sharing scheme. This, however, comes with a precision cost in that of only a few events can be monitored per short-time interval.

7.3 Noise reduction

The objective of the proposed characterization methodology is to produce a quantitative workload performance profile, which exposes the behaviour of the given workload in finer detail. However, the performance of an application can depend on a variety of known, and unknown factors. These factors can, and probably will, produce noise in the data, it is therefore important to profile the workload a statistically significant amount of times to reduce its impact. Doing several profiling runs will also help to remedy the precision loss due to hardware event multiplexing, as mentioned in Section 7.2.

Naturally, if the origin of the noise can be identified and eliminated, or is inconsequential, then the number of profiling runs can be lowered. In any case, performing profiling runs several times will allow us to more accurately capture the median, or average, performance behaviour of the given workload, in addition to allowing for the extraction of other statistical metrics on the resulting dataset.

Preset Events	Description
General performance events	
PAPI_FUL_ICY	Cycles with maximum instruction issue
PAPI_TOT_CYC	Total cycles
PAPI_TOT_INS	Instructions completed
PAPI_LD_INS	Load instructions
PAPI_LST_INS	Load/store instructions completed
PAPI_RES_STL	Cycles stalled on any resource
PAPI_SR_INS	Store instructions
PAPI_STL_ICY	Cycles with no instruction issue
Branch instruction events	
PAPI_BR_CN	Conditional branch instructions
PAPI_BR_INS	Branch instructions
PAPI_BR_MSP	Conditional branch instructions mispredicted
Level 1 cache events	
PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_ICM	Level 1 instruction cache misses
Level 2 cache events	
PAPI_L2_DCA	Level 2 data cache accesses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L2_ICA	Level 2 instruction cache accesses
PAPI_L2_ICM	Level 2 instruction cache misses
TLB events	
PAPI_TLB_DM	Data translation lookaside buffer misses
PAPI_TLB_IM	Instruction translation lookaside buffer misses

Table 7.1: Overview of PAPI preset events used for workload characterization.

Native events	Description
Runtime-related events	
Real cycles	Wall-clock elapsed time (in cycles)
Real usecs	Elapsed high-resolution real time (in microseconds)
Virtual cycles	Process virtual cycles
Virtual usecs	Process virtual time (in microseconds)
Wallclock usecs	Unhalted wallclock time (in microseconds)
Dynamic memory usage events	
Memory heap (at exit)	Memory heap size at process exit
Memory library (at exit)	Memory allocated to libraries at process exit
Memory locked (at exit)	Memory locked at process exit
Memory resident max (at exit)	Peak resident set size
Memory shared (at exit)	Memory shared at process exit
Memory stack (at exit)	Memory stack size at process exit
Memory text (at exit)	Memory allocated to code at process exit

Table 7.2: Overview of native events used for workload characterization.

7.4 Availability of workloads

There is a distinct lack of publicly available workloads/benchmarks pertaining to the edge computing paradigm [33], some reasons for which include that the research field of edge benchmarking is still in a nascent stage [32], privacy restrictions [23], or that existing benchmarks focus on specific platforms of application domains [30]. Varghese et al. [32] state that most edge benchmarks today rely directly on simulators or the data obtained from them, which runs contrary to the classic definition of benchmarking.

7.5 Generating synthetic workloads

Owing in due part to the current circumstances outlined in Section 7.4 regarding the availability of edge benchmarks, we have opted to use synthesised workloads for the purpose of experimentation. Additionally, synthetic workloads are easier to parameterize, and thereby allow us to study the proposed characterization methodology for a wider variety of workloads. For this purpose, we use the stress workload generator tool *stress-ng* [27], available for POSIX systems, which allows us to synthesize applications that stress a variety of system resources.

It is worth noting that the *stress-ng* documentation states that the tool is mainly

intended to make the machine work hard and trip hardware issues, and that it was never intended to be used as a precise benchmarking test suite. Nevertheless, the various stress tests that are available through *stress-ng* provides a good "jumping off" point for evaluating the characterization methodology presented in this thesis.

We will be using stress tests that predominantly stress either *cpu*, *file system I/O*, *memory (VM)* or *network* resources. Workloads that stress a combination of the mentioned resources are also generated.

7.6 Defining the class representatives

We follow the approach described in Section 5.1.1 to create the task shape of the class representatives. Specifically, we group the workloads based on the system resource they are designed to stress. Workloads generated using a combination of two or more stressors are excluded from these groupings. After creating the groupings, the individual workloads are profiled and then aggregated to create the data sets that represent the classes.

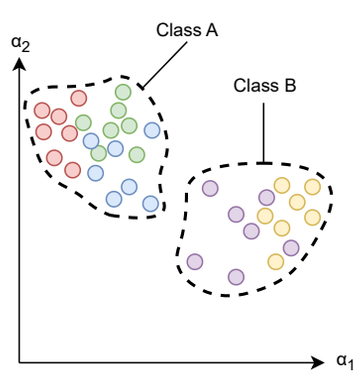

Figure 7.1: Workloads characterized by two arbitrary attributes α_1 and α_2. The workloads are grouped into class A or B depending on what resource they are designed to stress. The intuition is that applications with the same use case will have similar task shapes, and it should therefore make sense to group them together.

7.7 Attribute correlation metric

We use *Spearman's rank correlation coefficient* to measure the statistical dependence between attributes. The *Pearson correlation coefficient* is also commonly

used in the literature to check for correlations between attributes [8]. However, *Pearson's* assumes finite variance and finite covariance, assumptions that we cannot make on our attributes. *Spearman's* instead provides a measure of the monotic relationship between variables, and therefore makes no assumptions on the distribution of our attributes.

7.8 Similarity metrics

For the purpose of classification, We choose Euclidean distance (Eq. 7.1) to measure the similarity between task shapes. Specifically the distance between the task shapes of our class representatives and the task shapes of individual workloads in the attribute space. The advantage of using the Euclidean distance is that its simple to implement and fairly easy to interpret as opposed to the Manhattan distance.

$$d(p,q) = ||p - q|| = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{7.1}$$

In Eq. 7.1, p and q are two points in an n-dimensional Euclidean space.

In addition, we measure the Cosine similarity when performing classification but do not use it to compute the final classification scores. Rather, it will serve as a complimentary to the final classification whereby it either corroborates or refutes the result.

$$S_C(P,Q) = cos(\theta) = \frac{\mathbf{P} \cdot \mathbf{Q}}{||\mathbf{P}||\,||\mathbf{Q}||} \tag{7.2}$$

In Eq. 7.2, θ is the angle between the vectors $\boldsymbol{P}$ and $\boldsymbol{Q}$.

To compute the measurements in Eq. 7.1 and Eq. 7.2 we elect to use the greedy median, defined in Section 4.4.2, as the center point of our task shapes, as this center point is much faster to compute compared to the geometric median.

7.9 Classification scores

Owing to the fact that our data exists in a high-dimensional space, the distance measurements can become quite large in magnitude and can therefore be difficult to analyze. For this purpose we want the classification, or "degree of truth", score for each class to be relative to each other but also be weighted by their closeness to a point of reference. In this case the point of reference would be the center point of the task shape to be classified.

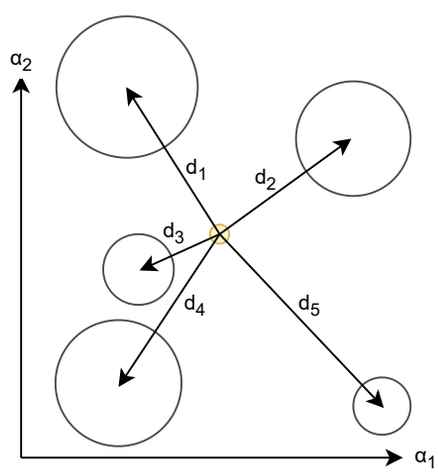

Figure 7.2: Measuring distances to the center point of each class representative from a point of reference. In this case, the point of reference is the center point of a workload represented by the yellow task shape. The class representatives are represented by white circles.

We can calculate such a score w_i using the following formulation:

$$w_i = \frac{1}{d_i * \sum_{j=1}^{D} \frac{1}{d_j}}, \quad i, j = 1, 2, 3, ..., D \tag{7.3}$$

Where d_i is the Euclidean distance (Eq. 7.1) from the reference point to a point i, and D is the total number of distances considered. the score w_i is calculated by taking the inverse of the distance d_i and multiplying it with the inverse of the sum of all inverse distances considered. We normalize the scores so that $\sum_i^D w_i = 1$. the final results can then be displayed as percentages, which is easier to interpret.

For the example given in Fig. 7.2, the total number of considered distances D is 5. If we compute the scores w_i in the example, we will find that the highest score is given to the class with distance d_3 from the point of origin, while the lowest score is given to the class which is a distance d_5 from that same point of origin.

8

Experimental setup

8.1 Testbed

The following equipment is used to conduct experiments:

OS	Linux 5.19.0-42-generic
CPU	Intel Core i5-8500T @ 2.10GHz
No. Cores	6
No. HPCs	10

Table 8.1: Hardware specifications

We perform all hardware event measurements with the PapiEx command-line interface extension for PAPI.

8.2 Parameterization of *stress-ng*

We construct a set of "*jobfiles*", which are sets of unique parameter configurations, that causes *stress-ng* to stress the system in various ways depending on which stress method, or stressor, is chosen. The majority of the constructed jobfiles are configured to only use one type of stressor.

Each jobfile is scheduled to time out after 10 seconds, with the exception of jobfiles pertaining to the VM stressors (Table 8.2). Instead, these jobfiles are scheduled to stop after a given amount of *bogo operations* (bogus operations per second) equating to roughly 10 seconds per VM stressor. This is done due to a quirk most likely caused by the interaction between *PapiEx* and *stress-ng* where the sub-processes that were spawned by the *stress-ng* VM stressors fail to time out when using the normal timeout parameter, causing them to run indefinitely.

We also configure the jobfiles such that *stress-ng* runs the configured stressor on all available processors on the system concurrently to eliminate the possibility of thread migration, which can potentially cause noise in the data.

8.3 Benchmarks

We generate 29 workloads in total, 23 of which are generated using only one type of stressor (Table 8.2). We refer to the *stress-ng* manpage and group them based on what part of the system the stressors are designed to stress.

Grouping	Stress method(s)
CPU	*afalg, branch, fft, jmp, mergesort, muladd64, sieve, matrixprod*
I/O	*direct, dsync, iovec, seek*
Network	*dccp, pingsock, udp, udp-flood*
VM	*bigheap, cachestripe, mlock, pageswap, read64, swap, write64*

Table 8.2: Overview of stress tests used to create single-stressor benchmarks. The stressors are grouped after which component of the system they are predominantly designed to stress.

One of the CPU-stressor workloads was generated using the "matrixprod" CPU stress method. According to the *stress-ng* manpage, this method is exceptionally good at stressing the CPU, and we therefore expect its performance behaviour to stand out among the CPU stressors.

The remaining five workloads (Table 8.3) are unique in that they were generated using a combination of stressors. Workloads that are generated using a combination of two stressors are scheduled so that there is an equal amount of cores allocated to each of them. I.e. the jobfiles are configured so that an equal amount of processors are tasked to run each stress test.

Further, for workloads generated using a combination of three or more stressors, we configure the jobfile to allocate cores unevenly among the stressors, such that one type of stressor is favoured over the others.

We also configure some of the multi-stressor workloads (e.g. "*CPU and I/O #1*", and "*CPU and Network #2*") to, in part, consist of "new" stressors that are not part any of the groupings in Table 8.2. To be more specific, both the "judy" and "correlate" stressors are designed to stress the CPU, but we keep them separate from the CPU grouping to see how our fuzzy classification model responds to new data.

Name	Stressors used & Core allocation
CPU and I/O #1	*correlate (3), iovec(3)*
CPU and I/O #2	*fft (3), seek (3)*
CPU and Network #1	*fft (3), udp(3)*
CPU and Network #2	*judy (3), dccp (3)*
I/O and Network	*seek (3), udp (3)*
CPU and Network and I/O	*matrixprod (3), dccp (2), seek (1)*

Table 8.3: Overview of benchmarks that combine two or more stressors. The numbers in parenthesis in the second column show the amount of cores allocated to each stressor.

The idea behind the multi-stressor workloads is to try to capture the performance behaviour of more complex real workloads that might exhibit characteristics that are indicative of two or more workload classes.

9

Evaluation

In this chapter we present and evaluate the result of experiments performed. First we present the results of characterizing the single-stressor and multi-stressor workloads using the hardware events listed in Tables 7.1 and 7.2, and perform a preliminary attribute reduction to remove attributes that have zero informational value. The reduced characterization is then presented and evaluated. Based on the result of characterization, we then construct our class representatives and classify the multi-stressor workloads using the proposed fuzzy classification model. The results of classification is then presented and evaluated.

9.1 Experiments performed

- We apply the methodology outlined in Chapter 6 to produce fine-grained characterizations of the workloads given in Tables 8.2 and 8.3. Each workload is profiled with PapiEx an number of times $R = 20$ to reduce the effect of noise (outliers, inaccuracy due to multiplexing, etc.) on the data. We instrument the HPCs to monitor the hardware events given in Tables 7.1 and 7.2. As stated in Section 8.2, all available cores $U = 6$ are scheduled to work concurrently for each profiling run, which adds up to $M = R * U = 120$ data points per workload.

- We perform a preliminary attribute reduction and remove attributes with

zero informational value before evaluating the results on a per group basis.

- After characterization, we collect the performance data of single-stressor workloads with similar performance behaviours into unified data sets to construct a set of class representatives, and evaluate the resulting classes.

- We test the classification accuracy of our fuzzy classification model by classifying the single-stressor workloads, with the expectation that the single-stressor workloads will be have the highest class score towards the class representative in which they are a constituent of.

- Further, we attempt to classify the multi-stressor workloads (Table 8.3) using the same model, here with the expectation that that the highest classification scores will be given to the classes of which the component stressors are a part of.

9.2 Preliminary characterization results

The full results of performing the characterization methodology on the single-stressor workloads (Table 8.2) is shown in Figures 1, 2, 3 and 4 in the appendix. The full characterization of the multi-stressor workloads (Table 8.3) are shown in Fig. 5 in the appendix. Owing to the high dimensionality of the workloads, we elect to show the results of characterization graphically as box plots along each attribute dimension. We exclude plots for the attribute *Memory locked (at exit)* from the full characterization, as its value across the entire set of workloads is zero.

9.3 Attribute reduction

We analyse the full characterization on the workloads and identify a set of attributes that, for our workloads, are not useful for characterization for various reasons.

9.3.1 Attributes with zero variance

Attributes with zero variance across the entire dataset are not useful for characterization as they have zero informational value with regards to differentiating the workloads. The memory related attributes *Memory text (at exit)*, *Memory*

locked (at exit) and *Memory library (at exit)* have zero variance for the entire set of data and can therefore be immediately excluded from the characterization.

The attribute *Memory stack (at exit) KB* varies discreetly between the values 460, 464 and 468 for each of the workloads, seemingly at random. The median value for each of the groupings is 464, except of the Network grouping where the group median is 468. However, three of the four workloads in the Network grouping independently report a median value of 464. We interpret these findings as equivalent to the attribute having zero variance, and conclude therefore that the attribute in question is not inductive to workload characterization in our case.

9.3.2 Attributes with perfect correlations

We inspect the box plots for each of the workload groupings to find attributes that have a perfect correlation between them. If two attributes are perfectly correlated, it is possible to infer one from the other and we therefore only require one of them.

A visual inspection of the *PAPI_L1_ICM* and *PAPI_L2_ICA* box plots indicates that there is a strong correlation between the two datasets for each of the single-stressor workloads. Indeed, the *Spearman rank correlation coefficient* between the two attributes for the entire dataset, including multi-stressor workloads, is $\rho = 1.0$, confirming that they are monotonically related. This can be explained by the fact that a *level 1 instruction cache miss*, leads directly to a *level 2 instruction cache access*. We therefore remove the attribute *PAPI_L2_ICA* as it can be inferred from the attribute *PAPI_L1_ICM*.

The the two attributes *Real cyces* and *Real usecs* are also identical for each of the groupings, with only a difference in magnitude. The reason for this is that *Real cyces* is calculated by taking the elapsed high-resolution real time (*Real usecs)* and multiplying it with the maximum operating frequency the processor can run at. The same pattern can be seen for the attributes *Virtual cycles* and *Virtual usecs*, and for the same reason as stated above. The Spearman rank correlation coefficient between the two attribute pairs are $\rho_{real} = 1.00$ and $\rho_{virt} = 1.00$ respectively. We therefore remove the two attributes *Real usecs* and *Virtual usecs*.

9.3.3 Real-time reliant attributes

Attributes that rely on real-world time, or wall clock time, such as *Real usecs*, *Real cycles* and *Wallclock usecs*, are limited by the scheduled timeout configured in the jobfiles. This means that the values reported for these attributes will be the same (around 10 seconds) for all of the workloads, and therefore provide no real informational value regarding workload characterization. The only exception to this is for the workloads in the VM grouping, where we see some minor magnitudal variation along the the mentioned attributes. This is because the VM workloads are configured to time out after a certain amount of *bogo operations*, instead of a set amount of time.

9.4 Reduced characterization results

The results of characterization using the reduced set of attributes is shown in Fig. 9.1 for the single-stressor workloads, and in Fig. 9.2 for the multi-stressor workloads. The attribute values are plotted on logarithmic scale to account for the large magnitudal differences between some of the workloads. We add solid vertical lines to the plots for the single-stressor workloads to more easily differentiate between the different workload groups.

9.4.1 Characterization of single-stressor workloads

In Fig. 9.1, we expected to see that the *matrixprod* workload had a higher magnitude for the majority of the attributes relative the other workloads in the CPU grouping, due its description in the *stress-ng* documentation. However, we see that this is only the case for attributes that relate to the level 1 and 2 data cache, i.e. *PAPI_L1_DCM*, *PAPI_L2_DCA* and *PAPI_L2_DCM*.

The attribute magnitude of the "*afalg*" workload is consistently small for a majority of the attributes, making it stand out from the rest of the workloads in the CPU grouping (Fig. 9.1). This discrepancy between "*afalg*" and the other workloads in the grouping is most clearly seen for the attributes *PAPI_TOT_CYC*, *PAPI_FUL_ICY* and *Virtual cycles*. The most likely reason for this has to do with the nature of the component the underlying "*afalg*" stressor is exercising, the AF_ALG socket. The purpose of the AF_ALG socket in Linux systems is to provide an access point to cryptographic services which can only be accessed from the kernel mode [17]. Since the workloads have been profiled in user mode, PapiEx is unable to measure the performance behaviour of operations that are being done in kernel mode and is therefore unable to accurately capture the performance behaviour of the workload.

For many of the attribute dimensions we see patterns indicating that workloads which belong to the same grouping have similar magnitudes (Fig. 9.1). An example of such patterns can be seen for the attribute *PAPI_FUL_ICY* where, disregarding a few outliers, The the workloads form "tight" upper and lower bounds on the region in which they exist along the given dimension.

Along other dimensions these regions are not as clearly defined, with relatively large inter-group variances between the workloads. Along the attribute dimension *PAPI_L1_ICM*, for example, the magnitudes of the various workloads vary a lot for the CPU, IO and VM groupings (Fig. 9.1). The CPU grouping exhibits a lot of variance along the *PAPI_L2_DCA* attribute dimension as well. In cases such as these, it could be beneficial to sort the workloads into more than one region so as to reduce the variance.

Focusing now on the differences and similarities of the workload groupings, we see that some groups are clearly distinguishable from the other groups for many of the attribute dimensions. Along the attribute dimension *PAPI_TLB_DM*, the VM group forms a distinct high-magnitude region, while the other groups form regions with similar upper and lower bounds (Fig. 9.1). An identical pattern can also be seen along the *Memory heap (at exit)* attribute dimension, when the VM grouping forms a clearly distinct region relative to the other workload groups.

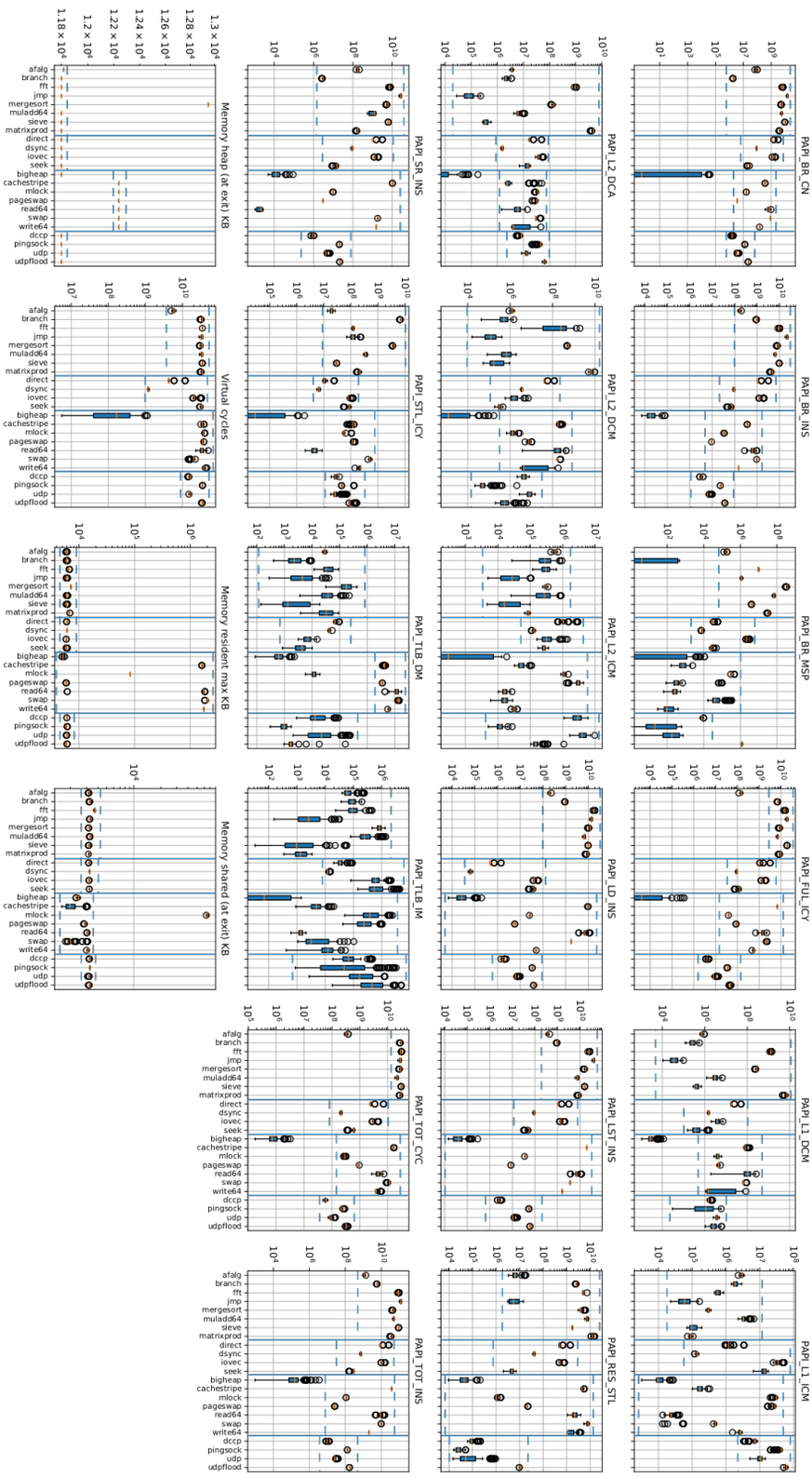

Figure 9.1: Characteristics of the single-stressor workloads after dimensionality reduction. Blue lines have been added in post to more easily distinguish between the workload groups. The dashed blue lines have been added to better visualize the upper and lower bounds of the groups along each attribute axis.

Similarly, along the *PAPI_TOT_CYC* and *PAPI_TOT_INS* attribute dimensions, the CPU grouping forms a highly distinct upper region compared to the other groups (Fig. 9.1).

9.4.2 Characterization of multi-stressor workloads

As the task shapes of the multi-stressor workloads are essentially combinations of stressors with different performance behaviours, the variances along each attribute dimension is quite large for each of the multi-stressor workloads. There are some exceptions to this, as can be seen for the attributes *PAPI_TLB_DM* and *PAPI_TLB_IM* in Fig. 9.2.

The multi-stressor workload "*I/O and Network*" has the least variance in general, indicating that its constituent stressors are similar in terms of performance behaviour. We also notice that the median values of the workloads, with the exception of "*I/O and Network*", are quite similar for many of the attribute dimensions. However, for the attributes *PAPI_L1_ICM*, *PAPI_L2_ICM* and *PAPI_STL_ICY* we see that the median values vary a bit (Fig. 9.2).

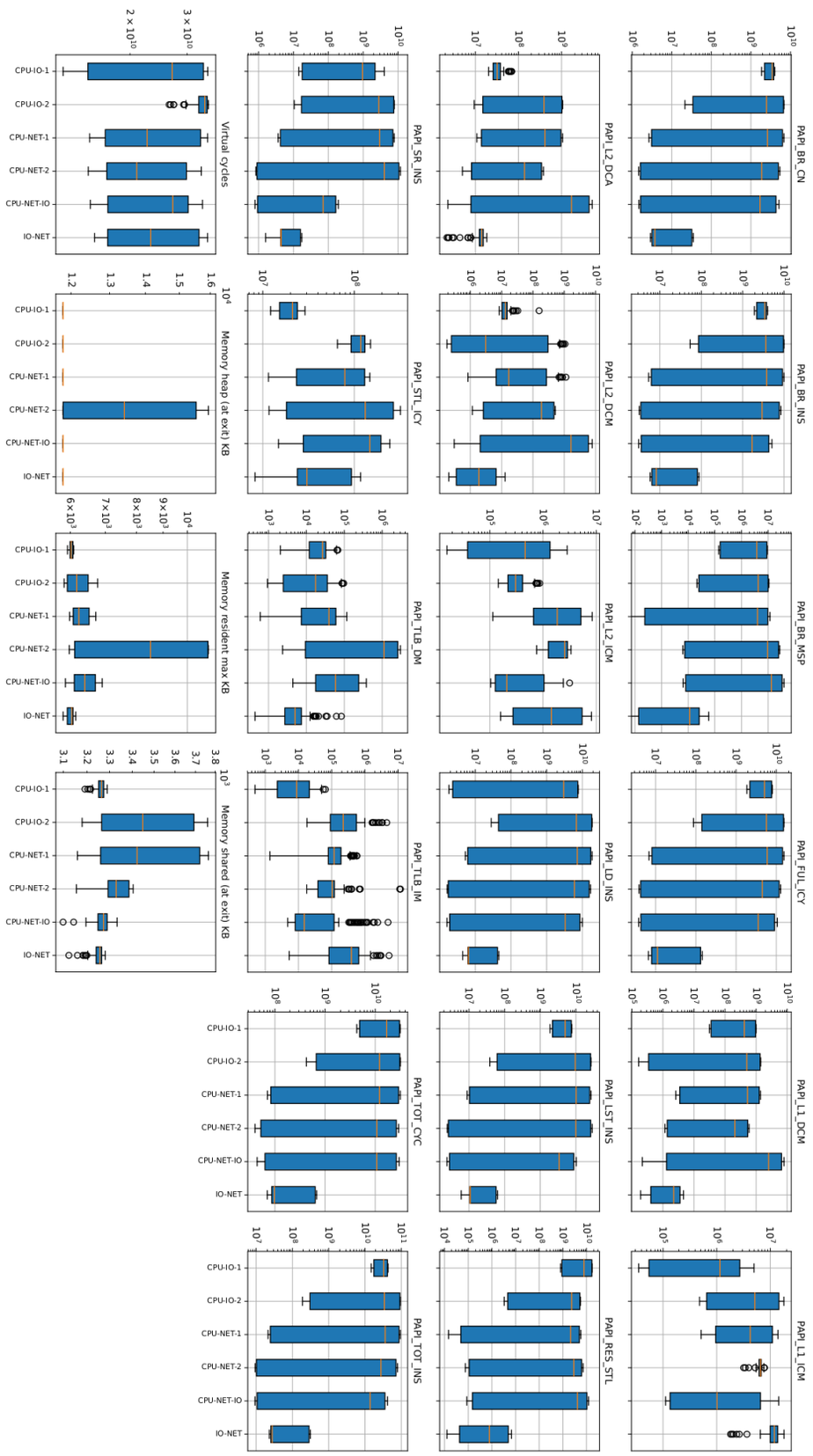

Figure 9.2: Characteristics of the multi-stressor workloads after dimensionality reduction.

9.5 Evaluation of class representatives

Motivated by the findings in Section 9.4, we begin collecting workloads into unified task shapes to define the class representatives. Since the data is multi-dimensional and fairly complex, we elect to use the same groupings as outlined in Table 8.2 to construct the unified task shapes. That is, we forego the step of further subdividing the classes based on attribute magnitudes, as outlined in Section 5.1.1, as this would require the usage of more intricate clustering methods that fall beyond the scope of this thesis.

The resulting unified task shapes are shown in Fig. 9.3, as logarithmic box plots along each attribute dimension. We see that the variances for each of the classes are relatively large for some of the attributes, while in others the variances are somewhat smaller.

The Network class has overall the smallest variances along each of the attribute dimensions, with the exception of the two attribute dimensions *PAPI_BR_MSP* and *PAPI_L2_ICM* (Fig. 9.3). This indicates that its constituent workloads are well suited to being grouped together. The same can be said for the IO class to a somewhat lesser degree.

The CPU class exhibits a large variance along the *PAPI_L2_DCM*, *PAPI_L2_DCA* and *PAPI_L1_DCM* attribute dimensions (Fig. 9.3). This is likely due to the inclusion of the "matrixprod" workload which reported higher magnitudes along these dimensions, as stated earlier.

The VM class reports the largest variances for each attribute overall, in Fig. 9.3, indicating that it is probably a badly defined class. I.e. The upper and lower bounds of the task shape along each attribute dimension are very far away from each other.

Focusing on the median value of the classes along each attribute dimension in Fig. 9.3, we see that for many of the attributes the reported median values are quite dissimilar across the classes. Examples of this includes the attribute dimensions *PAPI_BR_MSP*, *PAPI_TOT_INS* and *PAPI_RES_STL* where we can see that the classes are clearly discernible with respect to the median.

We can also see that the CPU class has the highest median value for 11 of the 22 attribute dimensions used for characterization, while the Network class reports the lowest median value for 8 of the attribute dimensions (Fig. 9.3). Incidentally, 7 of these attributes (*PAPI_BR_CN*, *PAPI_BR_INS*, *PAPI_FUL_ICY*, *PAPI_LST_INS*, *PAPI_RES_INS*, *PAPI_TOT_CYC* and *PAPI_TOT_INS*) are also ones where the CPU class has the highest median value.

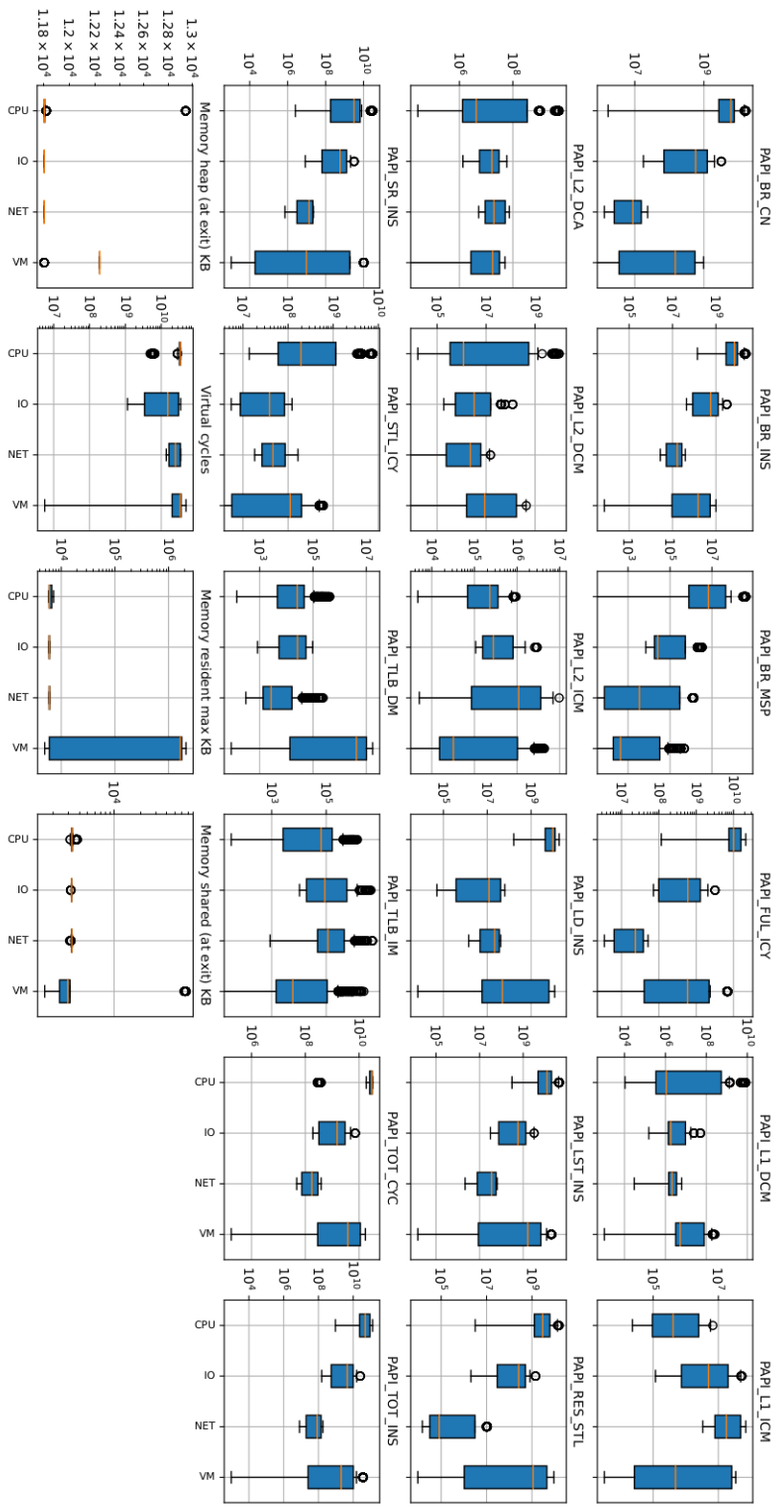

Figure 9.3: Characteristics of the class representatives after dimensionality reduction.

We also see in Fig. 9.3 that he median values for the IO class consistently falls in the middle relative to the other classes, never reporting median values that are significantly higher or lower than the others. The same pattern is seen for the VM class.

9.6 Single-stressor workload classification performance

We use the class representatives shown in Fig. 9.3 to apply our fuzzy classification model to the single-stressor workloads to get an indication of the classification accuracy. We expect that the classification score w_i (Eq. 7.3) will be the highest for the class representative in which each workload is a constituent of. We also calculate the Cosine similarity S_i (Eq. 7.2) for each pair of single-stressor workload and class representative, but do not consider it as part of the classification. That is, we only use it to potentially gain further insight into the resulting classification scores.

The classification results for the single-stressor workloads are shown in Tables 9.1, 9.3, 9.5 and 9.7, for the I/O, CPU, Network and VM stressor workloads respectively. In each table we highlight the column header for which we expected to see the highest classification, or truth, score w_i in green. We also highlight the highest classification score per row; green if the result aligns with the expectation, and yellow otherwise.

The corresponding Cosine similarity scores are shown in Tables 9.2, 9.4, 9.6 and 9.8 for the I/O, CPU, Network and VM stressor workloads respectively. Again we highlight the column header in which we expected the highest Cosine similarity value in green, and the highest Cosine similarity value per row in either green if correct, or yellow if wrong.

9.6.1 I/O workload classification performance

In Table 9.1 the results of applying our workload classification model on the I/O stressor workloads is shown. We see that for three of the four workloads the $w_{I/O}$ score is the highest, indicating that they are most similar to the I/O class representative. We can also see that all four of the workloads have the least similarity to the CPU class representative, reporting w_{CPU} scores between 3.8-9.3%.

The "*seek*" workload, even though it is a constituent of the I/O class represen-

Name	w_{CPU}	$w_{I/O}$	w_{NET}	w_{VM}
direct	9.1%	47.7%	24.8%	18.2%
dsync	9.3%	44.1%	27.2%	29.4%
iovec	6.6%	41.6%	29.6%	22.1%
seek	3.8%	12.7%	28.4%	55.1%

Table 9.1: Classification of the filesystem I/O stressor workloads.

Name	S_{CPU}	$S_{I/O}$	S_{NET}	S_{VM}
direct	0.956	0.682	0.423	0.500
dsync	0.863	0.964	0.835	0.877
iovec	0.822	0.984	0.882	0.916
seek	0.499	0.953	0.999	0.992

Table 9.2: Cosine similarity between each class representative and I/O stressor workload.

tative, has particularly low classification score towards the I/O class (Table 9.1). Looking at its characteristics in Fig. 9.1, we can see that for some of the attributes it reports a lower magnitude relative to its counterparts in the I/O grouping, which is what we are probably seeing being reflected in the classification scores.

Apart from the "*direct*" workload, all of the workloads in the I/O group report a high Cosine similarity value towards the I/O class (Table 9.2), indicating that the ratios between the attributes for each of the workloads are similar.

9.6.2 CPU workload classification performance

The classification accuracy for the CPU stressor workloads is very good, as can be seen in Table 9.3. Only two of the 8 workloads reports a higher score towards class representatives other than the CPU class representative, one of which ("afalg") we flagged earlier in Section 9.4.1 as having particularly small attribute magnitude values compared to the other CPU stressor workloads.

We can also see that the Cosine similarity corroborates the scores calculated by our fuzzy classification model (Table 9.4). The CPU stressor workloads generally report much higher magnitudes for many of the attributes relative to the workloads in the other groupings (Fig. 9.1), which might be the reason they are easy to correctly classify.

Name	w_{CPU}	$w_{I/O}$	w_{NET}	w_{VM}
afalg	7.4%	49.2%	26%	17.3%
branch	18.9%	24.4%	25.9%	30.5%
fft	43.8%	18.8%	18.2%	18.9%
jmp	35.2%	21.8%	21.2%	21.7%
mergesort	71.3%	9.5%	9.2%	9.8%
muladd64	63.7%	12.1%	11.6%	12.5%
sieve	42.8%	19.2%	18.6%	19.3%
matrixprod	49.9%	16.5%	16%	17.5%

Table 9.3: Classification of the CPU stressor workloads.

Name	S_{CPU}	$S_{I/O}$	S_{NET}	S_{VM}
afalg	0.651	0.992	0.978	0.988
branch	0.744	0.785	719	0.792
fft	0.966	0.600	0.335	0.422
jmp	0.913	0.515	0.237	0.320
mergesort	0.993	0.700	0.468	0.561
muladd64	0.989	0.742	0.507	0.591
sieve	0.964	0.598	0.325	0.411
matrixprod	0.966	0.734	0.528	0.621

Table 9.4: Cosine similarity between each class representative and CPU stressor workload.

9.6.3 Network workload classification performance

It seems the classification model struggles to classify workloads belonging to the Network stressor group (Table 9.5). That is, it does not give the highest classification score towards the Network class. However, we can see that all the workloads give a low score towards the CPU class, and that they interchangeably give high and low scores towards the I/O and VM class.

Name	w_{CPU}	$w_{I/O}$	w_{NET}	w_{VM}
dccp	4.1%	50.7%	30.9%	14.2%
pingsock	4.2%	12.8%	26%	56.8%
udp	4.1%	50.9%	30.7%	14.2%
udpflood	4.1%	12.7%	26.88%	56.2%

Table 9.5: Classification of the network stressor workloads.

Looking at the Cosine similarity scores in Table 9.6, we see that the similarity values for each of the workloads are more or less the same, with all of them reporting the highest Cosine similarity with the Network class. This indicates

Name	S_{CPU}	$S_{I/O}$	S_{NET}	S_{VM}
dccp	0.486	0.950	0.999	0.990
pingsock	0.490	0.951	0.999	0.991
udp	0.488	0.950	0.999	0.991
udpflood	0.492	0.952	0.999	0.991

Table 9.6: Cosine similarity between each class representative and network stressor workload.

that the workloads do belong to the same class, but that perhaps the Network class is ill-defined. I.e. The greedy median is not a good representative for the class.

9.6.4 VM workload classification performance

As seen in Fig. 9.3, the VM class reports a high variance for many of the attribute dimensions, which, as earlier stated in Section 9.5, indicates that perhaps the class could benefit from being further subdivided into two or more separate sub-classes. In Table 9.7, this is made even more apparent when looking at the classification scores.

Name	w_{CPU}	$w_{I/O}$	w_{NET}	w_{VM}
mlock	4.5%	13%	25.1%	57.3%
cachestripe	39.9%	19.7%	19.2%	21.1%
write64	4%	10.3%	17.5%	68%
bigheap	9.6%	42.9%	27.5%	19.8%
pageswap	3.8%	12.1%	25.4%	58.5%
read64	9.4%	22.1%	28.2%	40.3%
swap	10.9%	34.4%	28.1%	26.5%

Table 9.7: Classification of the VM stressor workloads.

Name	S_{CPU}	$S_{I/O}$	S_{NET}	S_{VM}
mlock	0.490	0.951	0.999	0.991
cachestripe	0.951	0.786	0.690	0.668
write64	0.575	0.970	0.991	0.999
bigheap	0.486	0.950	0.999	0.990
pageswap	0.496	0.952	0.999	0.993
read64	0.746	0.962	0.916	0.947
swap	0.872	0.862	0.735	0.812

Table 9.8: Cosine similarity between each class representative and VM stressor workload.

in Table 9.7, four of the seven workloads are classified as expected, with the highest score going towards the VM class, the other three instead report a higher similarity towards other classes. However, these three workloads ("cachestripe", "bigheap" and "swap") do give very similar classification scores towards the VM class, between 21.1% and 26.5%.

Only one of the workloads report the highest Cosine similarity score towards the VM class (Table 9.8). We can also see that the workloads consistently report a high Cosine similarity with the I/O class, but also with the VM class. This give a slight indication that the workloads in the VM grouping have similar ratios between the attributes, but not with the VM class.

9.7 Multi-stressor workload classification performance

We give our classification model the more complex objective of classifying the multi-stressor workloads (Fig. 9.2), with the expectation that the highest classification scores will be given to the classes of which the component stressors are a part of.

Again we calculate the Cosine similarity S_i (Eq. 7.2) for each pair of multi-stressor workload and class representative, but do not consider it as part of the classification.

The results of classifying the multi-stressor workloads is displayed in Table 9.9, and the Cosine similarities are shown in Table 9.10. The highest classification score or Cosine similarity value per row is highlighted in green if it matches either one of the two to three expected classes, and yellow otherwise.

Name	w_{CPU}	$w_{I/O}$	w_{NET}	w_{VM}
CPU and I/O #1	31.9%	23.2%	21.5%	23.3%
CPU and I/O #2	33.8%	21.5%	21.0%	23.6%
CPU and Network #1	33.2%	23.5%	21.2%	22.0%
CPU and Network #2	23.6%	27.7%	24.1%	24.6%
I/O and Network	2.0%	13.5%	73.8%	10.6%
CPU and I/O and Network	12.2%	28.0%	27.8%	31.9%

Table 9.9: Classification of the multi-stressor workloads.

Apart from the workload "*I/O and Network*", the classification scores do not favour one class over others to a relatively high degree (Table 9.9). We expected

Name	S_{CPU}	$S_{I/O}$	S_{NET}	S_{VM}
CPU and I/O #1	0.979	0.792	0.577	0.660
CPU and I/O #2	0.964	0.856	0.664	0.731
CPU and Network #1	0.983	0.726	0.489	0.568
CPU and Network #2	0.981	0.767	0.546	0.626
I/O and Network	0.487	0.950	0.999	0.990
CPU and I/O and Network	0.880	0.935	0.818	0.877

Table 9.10: Cosine similarity between each class representative and multi-stressor workload.

to see that classification scores for the multi-stressor workloads favoured, to a high degree, the two or three classes of which the constituent stressors were a part of. What we see instead is that the scores are more or less "evenly distributed" with only a small difference of about 3-10% between the highest and next highest scores per row, and a slightly larger difference of about 4-19% between the highest and lowest scores per row.

The classification scores for the "*I/O and Network*" workload is heavily favoured towards the Network class at 73.8%, with a score of only 13.5% towards the I/O class (Table 9.9) despite both constituent stressors having the same amount of allocated cores (Table 8.3). We saw in Table 9.1 that the classification model struggled to correctly classify the "*seek*" workload, owing in due part to the fact that it reported a relatively low attribute magnitude compared to its I/O stressor counterparts. Indeed, we can also see in Fig. 9.1 that this workload has a performance behaviour that is actually more similar to the Network stressors, which might be the reason we see that the classification score w_{NET} in Table 9.9 is considerably higher than the other classification scores for the "*Network and I/O*" workload.

Two of the workloads, "*CPU and Network #2*" and "*CPU and I/O and Network*", reports the highest classification scores towards classes in which neither of their constituent stressors are a part of (Table 9.9). The "*CPU and I/O and Network*" workload assigns the lowest score towards the CPU class, even though more cores were allocated to the CPU stressor component (Table 8.3). The might be an indication that the greedy median is not a good representative for the workloads.

In Table 9.10, each of the multi-stressors reports a high Cosine similarity value towards at least one of the classes in which their constituent stressors are a part of. Interestingly, all of the workloads that consisted of a CPU stressor, apart from "*CPU and I/O and Network*", reports the highest Cosine similarity value towards the CPU class. A similar pattern is seen in Fig. 9.9 for the mentioned

workloads.

10

Discussion

10.1 Workload characterization using HPCs: Feeding two (or more) birds with one scone

We find that, when applying our proposed characterization methodology to synthetic workloads, we are able to produce high-dimensional workload characterizations that are able describe the performance behaviour of the given benchmarks in fine detail. The resulting workload characterizations are also relatively straightforward to interpret, given intermediary knowledge about hardware events.

The proposed methodology can easily be applied to other untested workloads. The only requirement is having access to HPCs, and being able to monitor them. Most modern microprocessors have built-in HPCs, and they can be easily instrumented using widely available event-monitoring tools such as *PAPI* and *perf_events*.

One of the benefits of leveraging hardware performance events to characterize workloads is that not only are we getting the fine-grained workload characterizations we are after, but we also get to profile the performance behavior of the application in question. Even though we are mainly using the HPCs for the purpose of characterization, we can still use them for one of their intended purposes; conducting performance-analysis. Given an arbitrary edge computing application or task, we might, from the resulting characterization, see that it

places abnormal strain on a given system resource. We now get the opportunity to, for example, fine-tune this application to run more efficiently on our system.

Consider also the case where we have two or more applications designed for the same edge computing use-case, such as Mixed Reality. By characterizing these applications/tasks using the methodology proposed in this thesis, we can now make a comparison between the performance behaviour of the resulting characterizations and select the application that has the best performance on our hardware setup.

10.2 Comparing results gathered from different systems

The downside of using microarchitecture dependent attributes, such as hardware events, is that the resulting workload characterization will invariably be different from system to system, due to different underlying hardware. That is, the resulting characterization of our proposed characterization methodology is in large part dependent on specific properties of the system resources. The compute power of the CPU, the network throughput, the storage capacity of the TLB-, CPU-, and Data-cache, etc. These are all component properties that will affect the performance behaviour of the application being profiled.

The consequence of this is that workload characteristics measured on one system might not be relevant for other systems, if the properties of the system resources are very different. A solution to this could be to profile the workloads on a "generalized" edge computing system that uses hardware components that commonly see usage in the edge computing paradigm. Workload characterizations produced on such a system could then be used as a starting point for related research that uses slightly different hardware configurations.

10.3 A prototype classification model

In Section 9.6 we evaluated the classification accuracy of our "proof-of-concept" fuzzy classification model, and found that, for some of the classes (Tables 9.1 and 9.3), the results were relatively good. However, for other classes (Tables 9.5 and 9.7) we found that our model struggled somewhat to classify the workloads correctly. The same can be said for the multi-stressor workloads where we saw discrepancies between the expected and actual results (Table 9.9).

We still believe that our proposed classification model has some merits, such as classifying workloads using a fuzzy membership scheme, but parts of the model will require further work in order to improve its performance.

The immediate downside of measuring distances using only the Euclidean distance between each center point is that the variance along each attribute dimension is not taken into account. For example, if the center point of a task shape is, by definition of the Euclidean distance, far away from the center point of some class representative but well within the region that the class representative encompasses, then that task shape will have a low classification score towards that particular class. An alternative approach is therefore required in order to take such situations into account when calculating the classification scores.

Another issue is that the current model weighs, in effect, each attribute dimension equally when calculating the classification scores. Attributes with a lower variance should be weighted more heavily than attributes with a large variance, as the region in which the class representative exists along such attribute dimensions is much tighter.

The model is also heavily dependent on the choice of how to define the class representatives, as these are the "ground truths" on which the model bases its classification decisions. In Section 11.5 we mention some alternative methods that can be used to define the class representatives.

11

Future Work

11.1 Adding microarchitecture independent attributes

Currently, our task shapes consists solely of microarchitecture dependent attributes. However, the work of some authors suggest that such characterizations can lead to misleading or erroneous conclusions [34]. Indeed, there are certain properties of edge computing workloads that cannot be captured by hardware performance events alone. There are already several works in the literature where microarchitecture-independent attributes are used for characterization. Combining these two approaches when selecting the task shape dimensions might prove advantageous for future characterization of edge workloads.

11.2 Applying the methodology to real workloads/benchmarks

The characterization methodology presented in this thesis was evaluated using synthetic workloads. As has been stated my many authors before [28, 32], synthetic workloads often do not accurately reflect the characteristics of real workloads. Future work could involve applying the proposed fine-grained characterization methodology to the few existing real edge computing workloads, to

evaluate if the proposed characterization methodology is viable in a real-world scenario.

11.3 Curating performance event counters using high-power computing domain knowledge

The work presented in this thesis made use of a set of hardware events automatically curated by PapiEx for workload characterization. Our results showed that several of these attributes were not useful for characterization for various reasons. Additionally, some attributes might not necessarily be of interest when characterizing edge workloads. A deeper understanding of hardware events is therefore needed to select the *optimal* configuration of hardware events for edge workload characterization. We suspect that such knowledge could be leveraged from the high-power computing domain domain, where HPCs are often used to solve a variety of problems relating to low-level performance analysis [11, 2, 12].

11.4 Workload characterization based on sequential sampling

The values of the hardware events used for characterization in this thesis were collected at the end of application runtime. It could be beneficial to instead characterize workloads by sampling its performance behaviour at given time intervals during runtime.

For example, instead of scheduling the constituent stressors of our multi-stressor workloads (Table 8.3) to run simultaneously, we can instead schedule them sequentially, either one at a time or intermittently. If we then profile the workload by taking samples of the hardware event values at given time intervals, the resulting workload characterization might then show that the workload displays characteristics indicative of different workload classes depending on the given time interval.

11.5 Applicability of machine learning techniques

We explicitly state that our proposed classification model is only intended as a "proof-of-concept" model to classify the high-dimensional task shapes, and we suspect that other clustering and classification methods might be better suited of this purpose. Specifically, we believe that methods and techniques from the field of machine learning can be leveraged to classify workloads. Indeed, the attributes of our *attribute space* can easily been seen as *features* in a *feature space*, commonly used for pattern recognition, regression and classification problems in the machine learning domain.

Various clustering and machine learning techniques are already being used in the literature to group and classify workloads [22, 25, 34]. Most common is the vector quantization method *k-means clustering*, which would be well suited to create the class representatives needed for our "proof-of-concept" fuzzy classification model. This technique also has the added benefit of allowing the user to specify the desired number of clusters, to achieve the desired class granularity. The *k-means clustering* algorithm could essentially replace *method 2*, outlined in Section 5.1.2, or be used in conjunction with *method 1* (Section 5.1.1), to construct the class representatives.

11.6 Feature reduction methods

Many methods exist in the literature across various domains to reduce the dimensionality of data. Considering that our objective is to produce fine-grained workload characterizations, some dimensions of the task shape will inevitably be found to be redundant. Which was made apparent in Section 9.3 when we evaluated the attributes. More robust methods to reduce the task shape dimensionality could therefore be implemented to find the optimal selection of attributes.

12

Conclusion

In this thesis we proposed and developed a novel methodology to produce fine-grained edge workload characterizations, with the purpose of addressing the limitations of existing edge workload characterization techniques. We leveraged hardware performance counters (HPCs), used to monitor hardware microarchitecture events, to extend the task shapes of workloads to higher dimensions, thereby obtaining the desired fine-grained workload characterizations. Further, we performed a rudimentary dimensionality reduction to identify redundant hardware events, and did an extensive analysis on the effectiveness of such fine-grained workload characterizations.

We also developed and presented a "proof-of-concept" classification model to classify workloads using the obtained fine-grained workload characterizations. The proposed model moves away from classifying workloads as "one-of" and instead assigns them classification scores based on their similarity to "ground truth" class representatives. From results we found that further work is required to improve the performance of the model in terms classification accuracy.

We conclude by mentioning that the proposed fine-grained workload characterization methodology presented in this thesis opens up a lot of interesting avenues of research with regards to characterizing edge computing workloads.

Bibliography

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. “On the Surprising Behavior of Distance Metrics in High Dimensional Space.” In: *Database Theory — ICDT 2001*. Ed. by Jan Van den Bussche and Victor Vianu. 2001, pp. 420–434. DOI: https://doi.org/10.1007/3-540-44503-X_27.

[2] U. Andersson and Phil Mucci. “Analysis and Optimization of Yee_Bench Using Hardware Performance Counters.” In: Jan. 2005, pp. 179–186.

[3] Ahmet Cihat Baktir, Atay Ozgovde, and Cem Ersoy. “How Can Edge Computing Benefit From Software-Defined Networking: A Survey, Use Cases, and Future Directions.” In: *IEEE Communications Surveys & Tutorials* 19.4 (2017), pp. 2359–2391. DOI: 10.1109/COMST.2017.2717482.

[4] Daniel Barry, Anthony Danalis, and Heike Jagode. “Effortless Monitoring of Arithmetic Intensity with PAPI’s Counter Analysis Toolkit.” In: May 2021, pp. 195–218. ISBN: 978-3-030-66056-7. DOI: 10.1007/978-3-030-66057-4_11.

[5] Shirley Browne et al. “A Portable Programming Interface for Performance Evaluation on Modern Processors.” In: *The International Journal of High Performance Computing Applications* 14.3 (2000-09 2000), pp. 189–204. DOI: https://doi.org/10.1177/109434200001400303.

[6] Shirley Browne et al. “PAPI: A Portable Interface to Hardware Performance Counters.” In: 1999.

[7] Maria Calzarossa, Luisa Massari, and Daniele Tessera. “Workload Characterization Issues and Methodologies.” In: *Performance Evaluation: Origins and Directions*. Ed. by Günter Haring, Christoph Lindemann, and Martin Reiser. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 459–482. ISBN: 978-3-540-46506-5. DOI: 10.1007/3-540-46506-5_20.

[8] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. “Workload Characterization: A Survey Revisited.” In: *ACM Computing Surveys* 48 (Feb. 2016). DOI: 10.1145/2856127.

[9] Yanpei Chen et al. *Analysis and Lessons from a Publicly Available Google Cluster Trace*. Tech. rep. UCB/EECS-2010-95. EECS Department, University of California, Berkeley, June 2010. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-95.html.

[10] E. J. Cockayne and Z. A. Melzak. “Euclidean Constructibility in Graph-Minimization Problems.” In: *Mathematics Magazine* 42.4 (1969), pp. 206–208. DOI: 10.1080/0025570X.1969.11975961. eprint: https://doi.org/10.1080/0025570X.1969.11975961. URL: https://doi.org/10.1080/0025570X.1969.11975961.

[11] Josh Davis et al. “Characterization of Power Usage and Performance in Data-Intensive Applications Using MapReduce over MPI.” In: Mar. 2020. ISBN: 9781643680705. DOI: 10.3233/APC200053.

[12] Jack Dongarra et al. “Accurate Cache and TLB Characterization Using Hardware Counters.” In: *International Conference on Computational Science (ICCS 2004)*. Springer. Krakow, Poland: Springer, 2004-06 2004. DOI: https://doi.org/10.1007/978-3-540-24688-6_57.

[13] M.R. Guthaus et al. “MiBench: A free, commercially representative embedded benchmark suite.” In: *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*. 2001, pp. 3–14. DOI: 10.1109/WWC.2001.990739.

[14] Cheol-Ho Hong and Blesson Varghese. “Resource Management in Fog/Edge Computing.” In: *ACM Computing Surveys* 52.5 (2019), pp. 1–37. DOI: 10.1145/3326066.

[15] Hui Li, David Groep, and Lex Wolters. “Workload Characteristics of a Multi-cluster Supercomputer.” In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 176–193. ISBN: 978-3-540-31795-1.

[16] Ankur Limaye and Tosiron Adegbija. “A Workload Characterization for the Internet of Medical Things (IoMT).” In: *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2017, pp. 302–307. DOI: 10.1109/ISVLSI.2017.60.

[17] *Linux Kernel Crypto API User Space Interface Library, Purpose Of AF_ALG*. URL: http://www.chronox.de/libkcapi/html/ch01s02.html.

[18] Jonathan McChesney et al. *DeFog: Fog Computing Benchmarks*. 2019. arXiv: 1907.10890 [cs.DC].

[19] Asit K. Mishra et al. “Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters.” In: *SIGMETRICS Perform. Eval. Rev.* 37.4 (Mar. 2010), pp. 34–41. ISSN: 0163-5999. DOI: 10.1145/1773394.1773400. URL: https://doi.org/10.1145/1773394.1773400.

[20] *PAPI homepage*. URL: https://github.com/icl-utk-edu/papi.

[21] *PapiEx homepage*. URL: https://bitbucket.org/minimalmetrics/papiex-oss/src/master/.

[22] Joseph Pasquale, Barbara Bittel, and Daniel Kraiman. “A static and dynamic workload characterization study of the San Diego Supercomputer center Cray X-MP.” In: *Measurement and Modeling of Computer Systems*. 1991.

[23] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. "Obfuscatory obscanturism: Making workload traces of commercially-sensitive systems safe to release." In: *2012 IEEE Network Operations and Management Symposium*. 2012, pp. 1279–1286. DOI: 10.1109/NOMS.2012.6212064.

[24] Mahadev Satyanarayanan. "The Emergence of Edge Computing." In: *Computer* 50.1 (2017), pp. 30–39. DOI: 10.1109/MC.2017.9.

[25] Virendra Singh Shekhawat, Avinash Gautam, and Ashish Thakrar. "Datacenter Workload Classification and Characterization: An Empirical Approach." In: *2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS)*. 2018, pp. 1–7. DOI: 10.1109/ICIINFS.2018.8721402.

[26] S. R. Shishira, A. Kandasamy, and K. Chandrasekaran. "Workload Characterization: Survey of Current Approaches and Research Challenges." In: *Proceedings of the 7th International Conference on Computer and Communication Technology*. ICCCT-2017. Allahabad, India: Association for Computing Machinery, 2017, pp. 151–156. ISBN: 9781450353243. DOI: 10.1145/3154979.3155003. URL: https://doi.org/10.1145/3154979.3155003.

[27] *stress-ng: a tool to load and stress a computer system*. URL: https://github.com/ColinIanKing/stress-ng.

[28] Klervie Toczé et al. "Edge Workload Trace Gathering and Analysis for Benchmarking." In: *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*. 2022, pp. 34–41. DOI: 10.1109/ICFEC54809.2022.00012.

[29] Klervie Toczé et al. *Extended edge use case characterization*. Zenodo, Jan. 2022. DOI: 10.5281/zenodo.5782871.

[30] Klervie Toczé et al. "Towards Edge Benchmarking: A Methodology for Characterizing Edge Workloads." In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2019, pp. 70–71. DOI: 10.1109/FAS-W.2019.00030.

[31] Yehuda Vardi and Cun-Hui Zhang. "The Multivariate L1-median and Associated Data Depth." In: *Proceedings of the National Academy of Sciences of the United States of America* 97 (Mar. 2000), pp. 1423–6. DOI: 10.1073/pnas.97.4.1423.

[32] Blesson Varghese et al. "A Survey on Edge Performance Benchmarking." In: *ACM Comput. Surv.* 54.3 (Apr. 2021). ISSN: 0360-0300. DOI: 10.1145/3444692. URL: https://doi.org/10.1145/3444692.

[33] Blesson Varghese et al. "Challenges and Opportunities in Edge Computing." In: Nov. 2016. DOI: 10.1109/SmartCloud.2016.18.

[34] Lei Wang et al. "WPC: Whole-Picture Workload Characterization Across Intermediate Representation, ISA, and Microarchitecture." In: *IEEE Computer Architecture Letters* 20.2 (2021), pp. 86–89. DOI: 10.1109/LCA.2021.3087828.

[35] L.A. Zadeh. “Fuzzy sets.” In: *Information and Control* 8.3 (1965), pp. 338–353. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(65)90241-X. URL: https://www.sciencedirect.com/science/article/pii/S001999586590241X.

Appendix

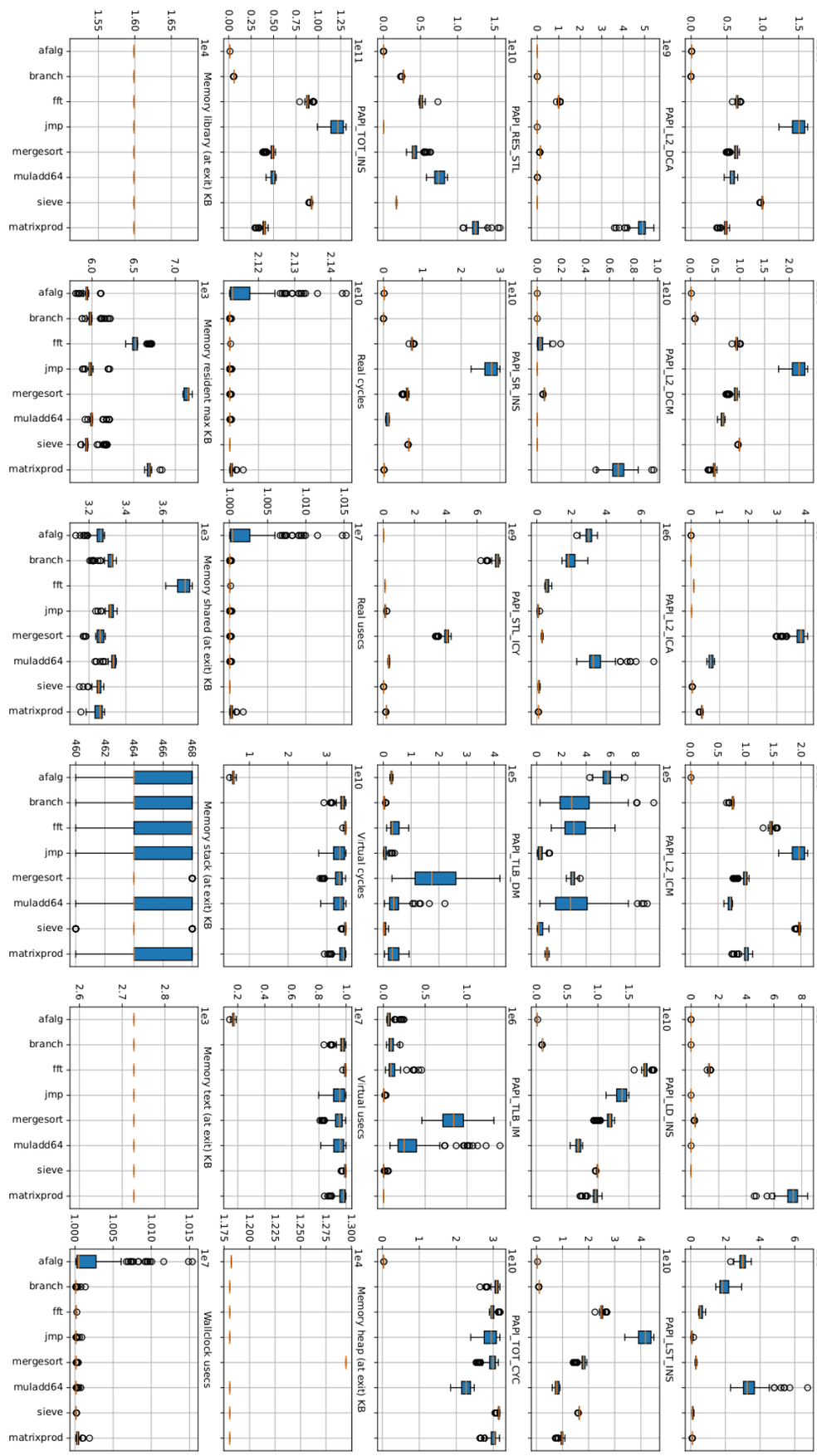

Figure 1: Full characterization of the CPU-stressor workload group.

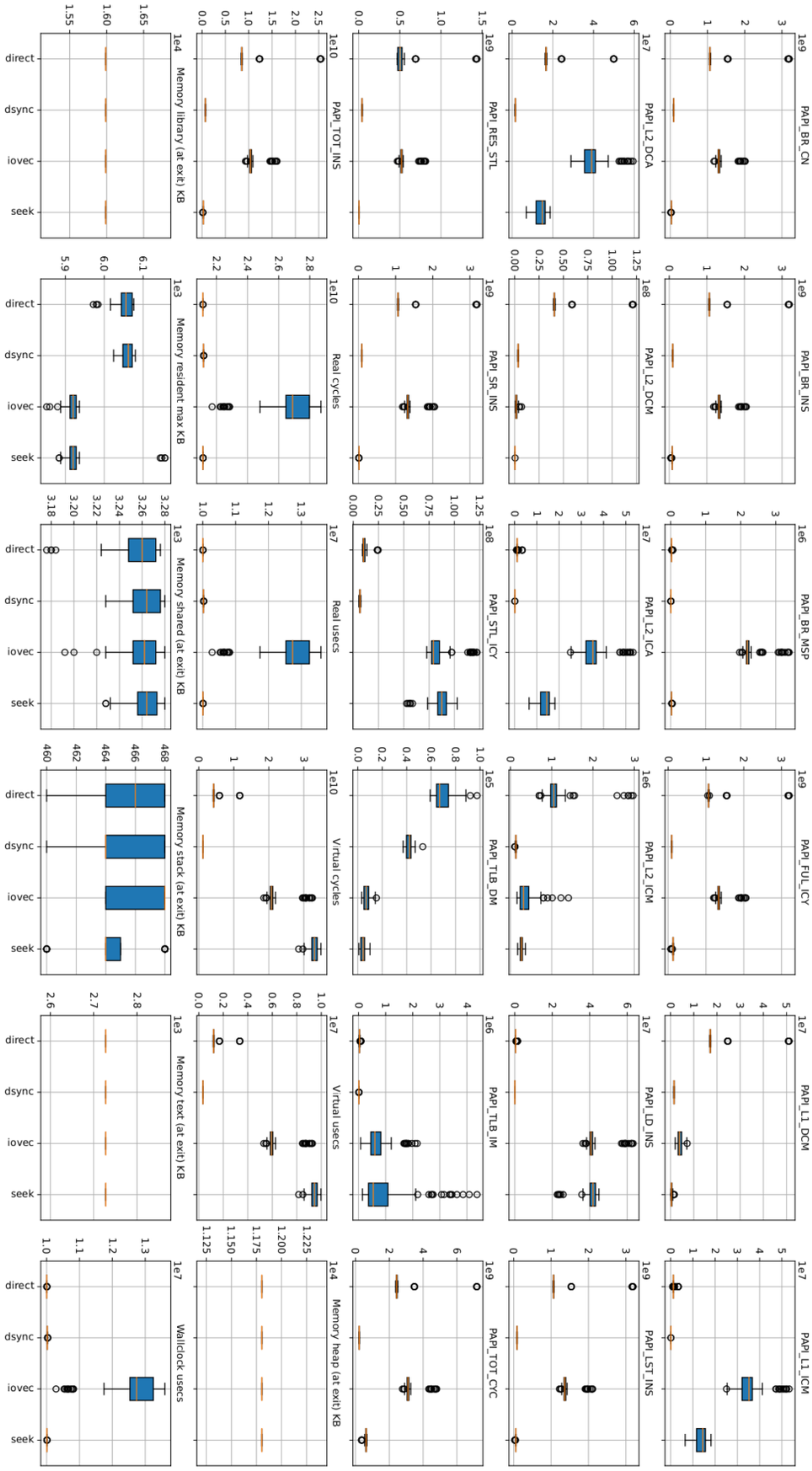

Figure 2: Full characterization of the filesystem I/O-stressor workload group.

Figure 3: Full characterization of the Network-stressor workload group.

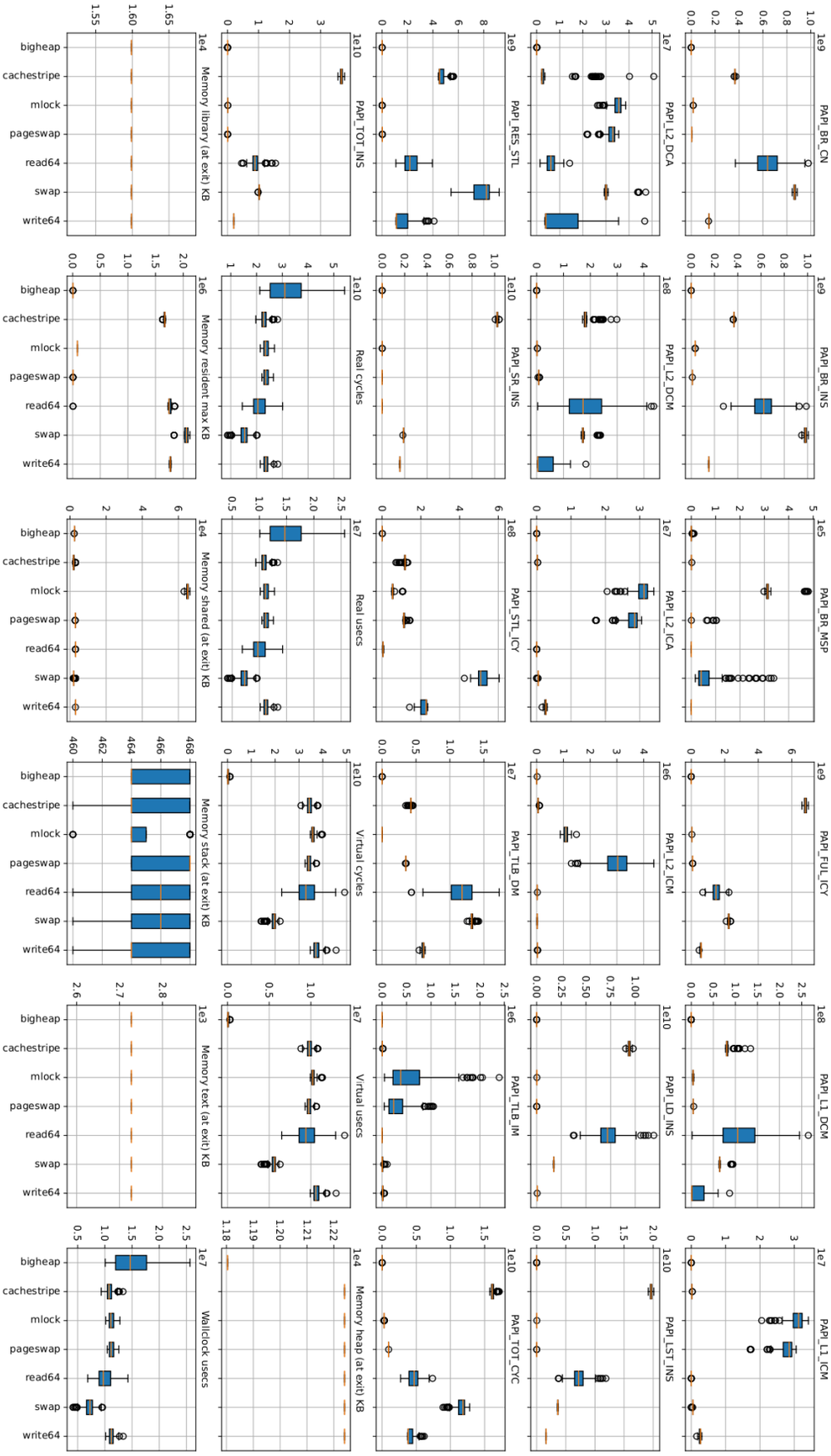

Figure 4: Full characterization of the VM-stressor workload group.

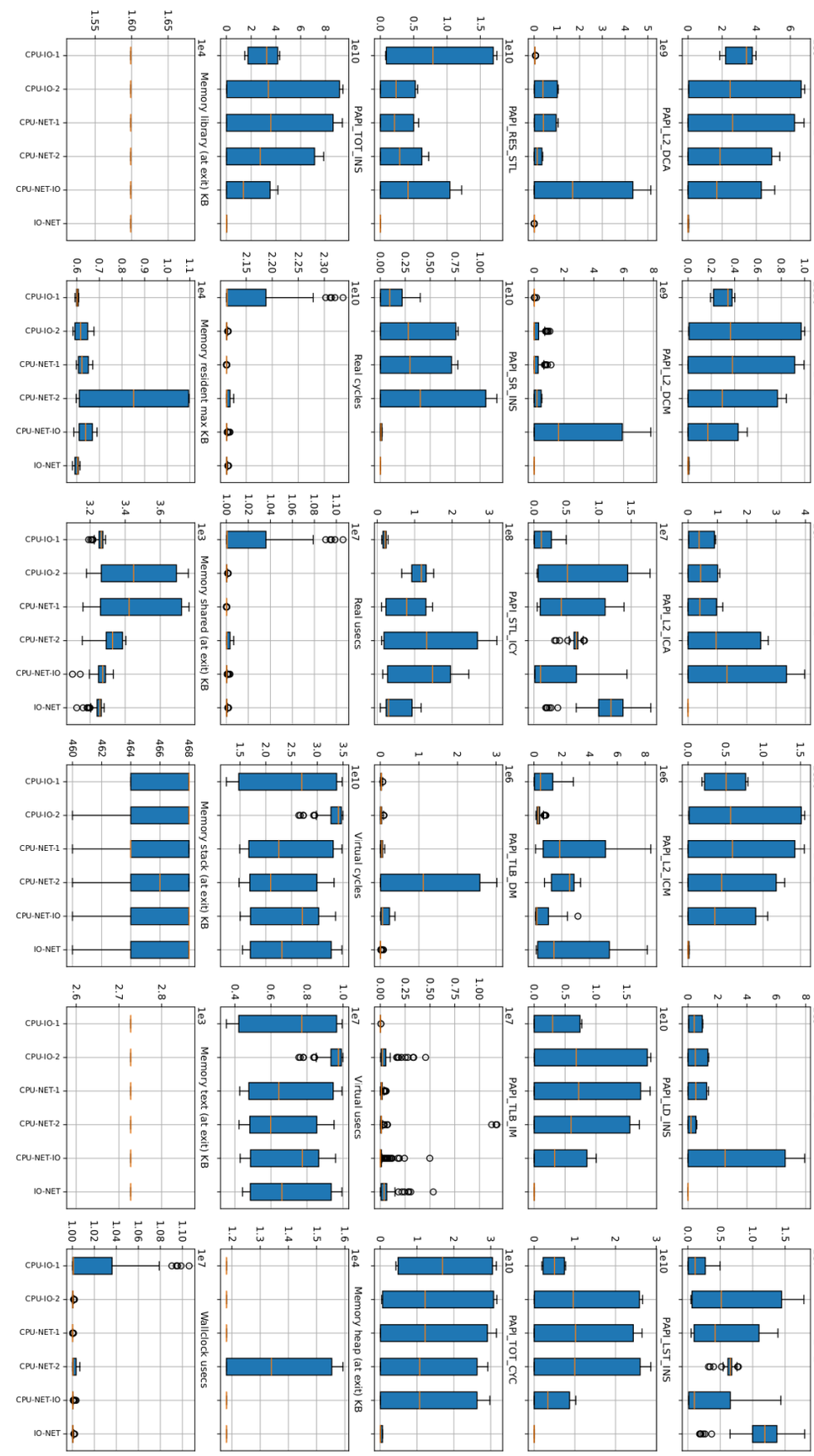

Figure 5: Full characterization of the multi-stressor workloads.