

Vrije Universiteit Amsterdam



VRIJE
UNIVERSITEIT
AMSTERDAM



Bachelor Thesis

LEGO, but with Servers:

Creating the Building Blocks to Design and Simulate Datacenters

Author: Jacob Burley (2599965)

1st supervisor: prof.dr.ir. Alexandru Iosup
external collaborator: ir. Georgios Andreadis
2nd reader: ir. Laurens Versluis

*A thesis submitted in fulfillment of the requirements for
the VU Bachelor of Science degree in Computer Science*

August 26, 2020

Contents

1	Introduction	5
1.1	OpenDC	6
1.2	Problem	7
1.3	Goal of this Work: Adding Prefabs to Datacenter Simulation	7
1.4	Research Questions and Approach	8
1.5	Main Contribution of the Work	9
1.6	Thesis Outline	10
2	Background	11
2.1	Viewing Datacenters as Topologies	11
2.2	Modularity in Computing	12
2.3	Domain-Specific Modules	13
3	Design of a Datacenter Hardware Representation	15
3.1	Why Prefabs?	15
3.2	Model for Datacenter Representation	17
3.3	Definition and Design of a Data Structure for Storing Prefabs	18
3.4	Designing Ways to Create and Interact with Prefabs	22
3.5	Applications of Prefabs	22
4	Implementation of a Prototype	24
4.1	Why Prototype?	24
4.2	Prototype Development	26
4.3	Lessons Learned	28
4.4	Future of the Prototype	28
5	Evaluation of Design & Implementation	30
5.1	Evaluation of the Design with Domain-Specific Prefabs	31
5.2	Evaluation of Prefab use in Practice	35
6	Conclusion	39
6.1	Our Answers to the Research Questions	39
6.2	Future Work	40
A	Appendix: Data from KLM Tests	46

List of Figures

1	Topology composition in OpenDC using prefabs	5
2	A conventional network-focussed datacenter topology	11
3	A simple topology in OpenDC	12
4	UML Class Diagram of Prefab Definition	19
5	A JSON representation of a CPU	21
6	Prototype showing the CLI operations it supports	24
7	Prototype demonstrating the import and deletion of a topology module	25
8	OpenDC Component Diagram	27
9	The starting position for evaluation	30
10	An example of a topology module for a Dell PowerEdge R440	33
11	Interactions required to complete a KLM task in two variants of OpenDC	36
12	Time taken to complete a KLM task in two variants of OpenDC	37

List of Tables

1	Relevant KLM operators used in our testing, and their times to complete	30
2	Market share distribution amongst server OEMs in Q1 of 2020	31
3	Overview of the HPC prefabs created for this thesis	31
4	Results of KLM testing before and after the addition of prefabs	46

Abstract

Tracking the rapidly growing consumption of cloud services, datacenters will likely increase in both number and capacity, and also improve their operational efficiency. Both of these developments will require numerous new datacenter designs. Because we are faced with a shortage of datacenter designers, it is important to make datacenter design accessible to a wider range of people, so that this growth may continue. To achieve this, we propose in this work to create datacenter designs with the use of pre-designed building blocks, or *prefabs*. We propose a hierarchical design for a data structure that allows for the representation of such building blocks. Additionally, we design the accompanying methods that enable us to perform operations on prefabs, when included in publicly available datacenter simulators such as OpenDC. We design a prototype to test the suitability of our designs, allowing us to carry out an implementation of prefabs into the codebase of OpenDC. We conduct small-scale but representative experiments based on the KLM method to assess the time taken to construct common datacenter layouts, and find that the prefab-based approach reduces the required time by a factor of 3.4, over the conventional approach not based on prefabs. We conclude that the addition of prefabs allows for a significant reduction in the number of interactions required to build datacenters in OpenDC. This reduction in interactions also thus leads to a reduction in the time taken to build datacenters in OpenDC. Our work enables a faster means of iteratively improving datacenter designs, while also enabling new users to get started with datacenter design much faster than they were previously able to.

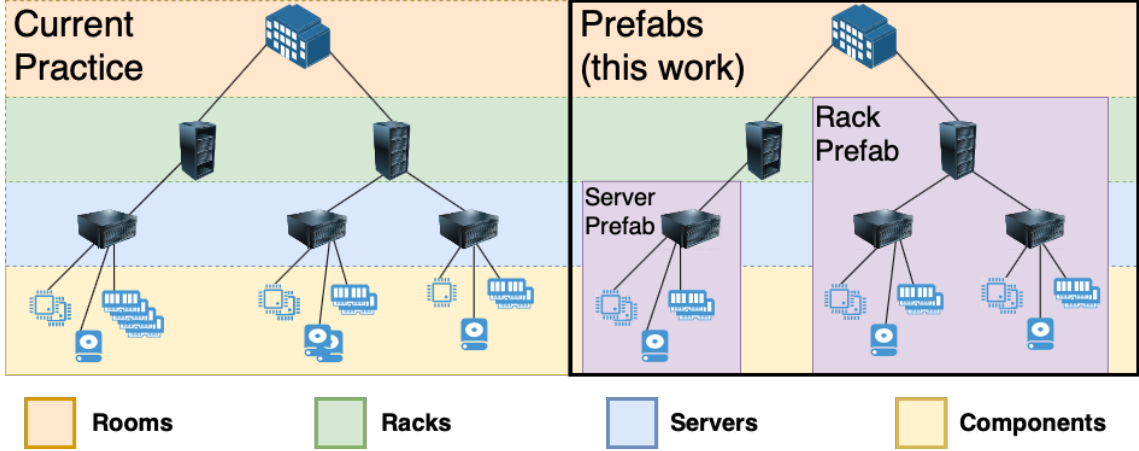


Figure 1: Prefabs in OpenDC compared against the current approach of composing topologies. Instead of building every datacenter from scratch, as in current practice, we propose to build from prefabs of several components each. This facilitates rapid prototyping of datacenters and sharing of prefabs.

1 Introduction

Corresponding to the rapid expansion in the use of cloud services by both consumers and businesses [23][27], the datacenters that house these services must also grow. Although many consumers may not ever see the cloud in terms of its underlying hardware, they rely on this complex infrastructure every day. In short, cloud services are hosted on servers, stored in datacenters. The configuration of such servers varies significantly, depending on which services need to run and on how many of each type of service run at once (the *workload*). Relational databases, for example, can have large memory requirements [22], so servers dedicated to running database services correspondingly must include large amounts of memory. Likewise, file servers may not need a particularly powerful CPU, but do require a lot of disks, and often also a network card with high maximum throughput [13].

All modern datacenters are designed; often, the largest datacenters are based on innovative, even unique designs. When architecting datacenters, many stakeholders must come together and agree on the various specifications of the datacenter, each with their own unique set of requirements. Stakeholders include the operators, potential and actual users, experts in various fields, and even representatives of the society. They each have their own needs and concerns, coming together to form a broad palette touching every complex problem of their time; for example, today datacenters consume about 1% of the global electricity consumption, making them a large consumer and thus an issue related to climate change [26].

As requirements become clear, system architects will try to design a system that meets these requirements. Designing a datacenter is not as easy as buying servers, placing them into an empty building, and putting them into use. Instead, designing datacenters is often a complex problem, for which no validated analytical models exist. Although some academic institutions often being forthcoming with the technologies used in their datacenters [30], there is no one-size-fits-all approach for datacenter design, making it

hard to determine which hardware can support a given set of workloads. For example, constraints such as budget or physical space often influence design decisions, with smaller (metropolitan) datacenters having less floor space and consequently making use of higher-density compute nodes, such as blade servers; but these servers consume more energy per square meter, and can cause more damage to the workload when they crash. Such trade-offs are typical of design problems in the field.

The shortage of up to 900,000 IT specialists by 2020 [15] paired with the growing demand for datacenters, and thus datacenter architects, presents a significant problem. This raises the question: how can we continue to support rapid growth in this sector, while the job market to support it does not grow to match? This research aims to extend existing datacenter simulation tools to allow them to be accessed by users with a less technical background, enabling a broader audience of people to design datacenters.

1.1 OpenDC

Due to the financial, space, and power requirements of server hardware, research into datacenter performance is more frequently being carried out with datacenter simulators. Such simulators provide the ability for researchers, and potential datacenter operators, to perform large-scale performance testing without having to make any capital expenditure in the process. In this way, prototyping becomes significantly cheaper, allowing for a much more flexible approach to datacenter research [18].

We focus in this work on OpenDC, an open-source discrete-event simulator that can be used to analyze the performance of workloads on massive computer systems [18]. Its purpose is to assist in performance testing of datacenters designed by users, with the intention that the information gained can be factored in when these systems are being designed, tuned, or compared with other systems for selection purposes. OpenDC simulates workloads using traces from the Grid Workload Archive [19] and other similar sources of real-world datacenter traces (e.g. the Workflow Trace Archive [35]), which are provided by datacenter operators or organizations who run workloads in datacenters, and are available to anyone through open-access. These traces provide metadata about the scheduling requirements of jobs frequently sent to datacenters, such as CPU usage throughout a job. Using this information in combination with information about the hardware infrastructure, OpenDC can simulate both the speed with which a workload will complete, and the resource usage throughout the execution of the workload. Thus, these simulations can assist businesses and educational institutions in highlighting and addressing performance concerns in their next-generation environments before any hardware has even been purchased.

Other discrete-event simulators also exist, such as BIGSIM [40], GridSim [6], SimGrid [8], and CloudSim [7]. Among these datacenter simulators, OpenDC claims to have a broad range of performance-related features [2], unique features related to common but thus far less studied phenomena that affect datacenters (e.g., performance variability [20][33] and correlated failures [39][21]), and a comprehensive library of resource management techniques. As a result of these characteristics, OpenDC can express a wide range of possible datacenter scenarios [36][2][37][34].

1.2 Problem

In its current state, OpenDC (and other simulators in the field) can be used to simulate user-specified workloads on user-designed systems. To create these simulations, users define a scenario in which the datacenter would be used, and specify each individual component used to build out an entire system. Effectively, users are doing all of the design work themselves, step-by-step. This requires a high level of technical understanding, and relies solely on the user’s knowledge of the intended workload to make component choices, as the user is responsible for choosing the hardware suited for that scenario. Moreover, even for the rare users with the technical skill, the number of detailed operations done manually, either in code or in the visual interface, increases the risk of mistake; and such building processes do not benefit from good, standardized validation practices.

Additionally, designing these datacenters is a time-consuming process, because there is no functionality to support the exporting or copying of existing designs, or parts thereof. This means that even if a user has a defined scenario in which they want to use their datacenter and a design for that scenario, any reuse of components designed previously requires them to be completely recreated. Similarly to the argument about designing from scratch, because the possibility of using a pre-existing design as a basis for a new design does not exist, non-technical users are faced with a steep learning curve when creating and experimenting with their own designs. Also, even expert users could make mistakes when trying to recreate each component, whereas with copying functionality such mistakes could be avoided entirely. (The lack of copying functionality is not necessarily surprising. Even large technology companies with savvy engineering teams have created products for mass adoption that lacked copying functionality at launch: an example of this being Apple only adding copy and paste features starting with the iPhone 3GS, 3 years into the products life¹.)

1.3 Goal of this Work: Adding Prefabs to Datacenter Simulation

We envision that prefabricated components (*prefabs*), that is, complete collections of components intended for a specific workload, could be “dragged and dropped” into a datacenter design, allowing for accelerated prototyping. In this way, hardware choices could be made based on which workload(s) the user wants to run. As a result, all a user would need to know to begin designing a datacenter would be its intended purpose, the workloads it would be required to run, and some basic compositional rules for prefabs when trying to assemble more complex or advanced configurations. A visualization of this approach can be seen in Figure 1.

We also envision that users could export the topologies that they design as prefabs, allowing for easy replicability of existing designs. The ability to save their designs as prefabs could support the sharing of prefabs within communities, allowing others to benefit from the sharing of knowledge regarding hardware requirements for different workloads, and promoting attribution of design efforts through an author property on the prefab itself. Overall, we hope that the addition of such functionality would result in a collection of prefabs that covers a wide variety of workloads, across an even wider variety of hardware

¹iPhone 3GS entry in Apple’s Iconic iPhone: A Visual History by Eric Griffith, <https://www.pcmag.com/news/apples-iconic-iphone-a-visual-history>

configurations.

Prefabs could provide further benefits when working with scalable workloads. If the workload relies on homogeneous nodes, these nodes could be defined as prefabs, and added repeatedly into the design as needed for capacity testing. Prefabs could consist of a single server, but could also scale up to a rack full of servers, or even a room full of equipment. This approach would enable users reliant on homogeneous hardware configurations to save their standard configurations as prefabs, further saving time when they need to add more of the same kind of node in new designs.

1.4 Research Questions and Approach

In this thesis, we aim to improve the ease of use of discrete-event simulators, taking as example OpenDC, through a prefab-based, easily expandable representation of datacenter hardware that allows for technical descriptions of hardware with a high degree of fidelity. In doing so, we answer the following three questions:

RQ1: *How to design a prefab abstraction that describes important technical and composable features of datacenter hardware to a high degree of fidelity?*

It is important for our vision of prefabs to be able to store them persistently. To support this goal, we need to create a data structure that can contain a high-fidelity hardware representation, along with its relevant metadata (such as tags, author, creation date, etc). For the purposes of this research question we define fidelity as how closely the model captures elements of reality.

Although smaller topologies may be expressed quite easily, we must consider that our solution should also scale to massive and complex topologies, while still encoding complete detail. It must also support a variety of operations to add, remove, and modify data stored within it to facilitate users working with it. This data structure should also be relatively straight-forward to understand and easily extensible, allowing it to be further expanded by open source contributors.

Our approach begins in Chapter 3, with understanding the shortcomings of the current implementation of topologies in OpenDC. Next, we seek to understand the components that are important to represent in the data structure, and identify components we no longer need to represent. From here, we examine a variety of common methods of representing data, and assess their suitability towards our desired way of representing datacenter hardware. Following on from this, we present a data structure design, and the decisions made during the design process, validating our design by using it to represent servers currently offered by popular *Original Equipment Manufacturers* (OEMs) in Chapter 5. We also assess what interactions need to be supported on the data structure, and ensure that the data structure design supports these interactions.

RQ2: *How to implement a prototype of the prefab abstraction?*

Prototyping the prefab abstraction is important for the addition of prefabs, as we may use new technologies that need to be tested and evaluated for compatibility with the current OpenDC software stack. We can use prototyping to become more familiar with these new technologies, analysing their strengths and potential shortcomings.

We also may use prototyping to determine what is and is not useful in our initial data structure design, and iteratively improve our design with these findings.

OpenDC is quite a large and complex project, with a lot of moving parts. In the process of adding a new data structure throughout OpenDC, it is important to continually test the code we are adding to the codebase for functionality. As a result, we must ensure that our prototype is representative of the structure of the current OpenDC project. Then, we can add the functions and data structure to our prototype, and evaluate whether the interactions we have designed are suitable for further integration into the OpenDC codebase. We can also use the prototype to determine if any additional changes need to be made to the existing codebase to support our extensions, and whether any of these modifications may adversely impact existing functionality.

In Chapter 4, we detail the process of designing and creating a prototype. We also provide a summary of the lessons learned during prototyping, and describe how we can use what we have learned to further improve our implementation of prefabs into OpenDC.

RQ3: *How to assess the use of prefabs in practice?*

In addition to implementing prefabs, we also should show that they are important to the OpenDC project, and to our goal of improving the usability of OpenDC. To do this, we should assess the impact that the addition of prefabs has on performing certain tasks in OpenDC, and determine what the benefits of prefabs are in concrete terms.

To do so, we must devise a testing strategy to show the improvements that our prefab implementation brings. We must also decide which metrics are important to measure when showing these improvements, and how we can measure these quantities.

In Chapter 5, we detail our experimental setup for objectively testing the improvement that prefabs bring to OpenDC. We also describe and discuss our experimental results, providing a short conclusion of what these results mean for the future of prefabs in OpenDC.

1.5 Main Contribution of the Work

This thesis pioneers an approach to representing datacenter hardware within a datacenter simulator. The main contributions of this thesis are:

1. We envision the use of prefabs for the design and simulation of datacenters.
2. We design of a data structure for storing datacenter hardware representations that is also easily usable and expandable, and an evaluation of said design in terms of the suitability of the data structure for purpose.
3. We implement this design into the main codebase of OpenDC, and an evaluation of this design comparing usability to the previous version of OpenDC.
4. We develop a library of components intended for High Performance Computing for use within OpenDC, built using the design specified in this thesis.

1.6 Thesis Outline

The remainder of this work is structured as follows. Chapter 2 provides more background on how modularity is currently used in computing, both in datacenters and in fields such as software engineering. In Chapters 3 and 4, we design a prefab data structure, reason about the choices that have been made during the design process, and touch on the process of implementing our chosen data structure into the OpenDC codebase. Chapter 5 provides an assessment of the suitability of the data structure and technologies we chose, now that they have been implemented in OpenDC. We assess both the implemented data structure, and the implemented methods of interacting with the data structure, using a different testing methodology for each. Finally, we discuss both the conclusion and future of this endeavour in Chapter 6.

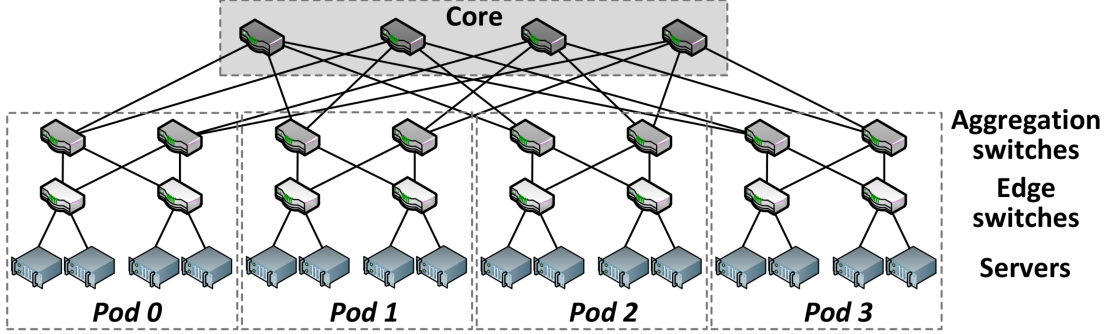


Figure 2: A conventional network-focussed datacenter topology [9].

2 Background

Servers today are used for a wide variety of tasks, ranging from hosting video games all the way up to calculations for nuclear research at the European Organization for Nuclear Research (CERN) [1]. As the need for servers grows, in part due to a massive increase in the demand for cloud services [29], the size of datacenters will also have to grow. As a result, datacenter operators will have design and organise these new datacenters into a logical organisational structure.

2.1 Viewing Datacenters as Topologies

In general, individuals working with datacenters think about their datacenter in terms of its topology. Often, this takes the form of a network topology [9], where a spanning tree is built from all of the participating nodes. An example of such a topology can be seen in Figure 2, which shows a conventional network topology in a datacenter. This topology is split into two sets of elements: the *core*, and the *pods*, where each pod contains two pairs of redundant servers. The purpose of this topology is to show a method of interconnecting each pod (as well as the servers within them), through redundant networking hardware. Each link represents a direct network connection between two elements of the topology. The links within this topology have been carefully designed such that the removal of one (or even several) links due to hardware failures (or other service interruptions) does not cause a loss of connectivity between servers. In this topology, there is no strict hierarchical relationship between elements of the topology. With the exception of each individual pair of servers, each of which is only connected to one edge switch, each element is connected to multiple other elements that can be viewed as its parents. As a result, each parent is connected to multiple children. This *many-to-many* design is key to the success of the design, ensuring duplicity among links.

For OpenDC, however, we explore a *one-to-many* relationship between components, increasing the importance of a components place in the hierarchy. For example, we can consider a chassis residing within a given rack to be a child of that rack, with good datacenter practice requiring that an in-use chassis is not kept outside of a rack. By defining these inter-component relationships, we can build our own spanning tree model, which allows us to easily manipulate parent objects and all of their children by simply

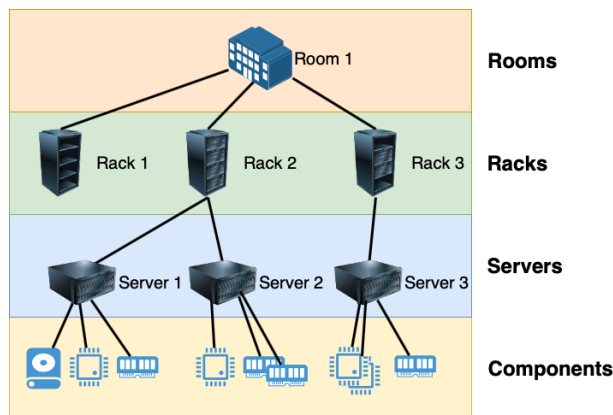


Figure 3: A simple topology in OpenDC.

moving the parent object elsewhere in the tree, reshaping our topology.

2.2 Modularity in Computing

Our concept of prefabs draws from a long tradition of modularity in computing. In this section, we cover *libraries* and package management in software engineering, tiered services in cloud computing, server designs in enterprise hardware, and brick-and-mortar modularization in the housing of datacenters. Several of the approaches taken to these problems use solutions that we find to have desirable characteristics when considering the usage of prefabs.

In software architecture, it is becoming increasingly common to rely on frameworks and libraries written by others in the software engineering community. The programmer often only uses one or two functions from this library, and does not necessarily understand how these functions work (nor do they need to). It is usually not necessary to understand the full workings of such a library, as the benefit comes from its utility in meeting certain requirements. We can view these software modules as prefabricated software, which an individual may add into their project with ease. Such a module may be added manually, by copying the relevant source files into the project, or may be added by means of a package manager.

npm is a package manager for **node.js** that seeks to make installing dependencies much easier for developers [38]. For open-source projects a developer can include a file listing all of the external packages they used in their project, and the specific versions of each of these packages, so that another individual can easily run the authors software on their own machine. **npm** also enables package authors to publish their packages in an online registry, so that other developers may download and use them. It follows, then, that it would be helpful to be able to describe datacenter hardware, and share these descriptions, in a similar way.

With the emergence of *Platform as a Service* (PaaS) and particularly *Software as a Service* (SaaS) offerings from major cloud providers, developers can implement entire layers of their software stack with just a few clicks. These layers still run in virtual machines, but the developer does not need to concern themselves too much with building or maintain the hardware or operating system. This concept is already offered by some

cloud providers: DigitalOcean is a *Virtual Private Server* (VPS) provider that harnesses a “marketplace” of pre-configured VPS templates (known as “1-Click Apps”) to simplify use of the platform [12]. Customers can then easily add a pre-configured VPS to their environment, without having to worry about operating system installation, configuration, or maintenance. Templates provided through the platform are typically created by the vendors of the software used in each template, and are thus of high quality and suitability for the chosen application.

In the realm of hardware, we are also seeing modularity emerge. Hewlett Packard Enterprise (HPE) produces a scalable server design, whereby more compute (in terms of CPU, Graphics and Memory) can simply be added as necessary [4]. A single operating system instance runs on this hardware, which can (at the time of writing) be expanded with up to 4 nodes, each containing 32 sockets supporting up to 28 physical cores per socket. It is thus possible to begin with a single system, and simply add “more of the same” when expanding datacenter capacity.

At a more macro level, Schneider Electric provide prefabricated, modular structures to house, power, and cool datacenters [31, 32]. While the offering from Schneider Electric does not include recommendations for actual compute hardware, IBM offers the possibility of a completely modular datacenter [17]. Such a datacenter can quickly be scaled up or down depending on the needs of the customer, using pre-designed components for power, cooling, and floorspace. These prefab datacenters are real-life examples of how datacenter design can be modular.

2.3 Domain-Specific Modules

Datacenters have many intended uses, depending on the intended *domain* of the datacenter, such as High Performance Computing (HPC), scientific computing, business-critical applications, generalized cloud infrastructure, etc. The design of datacenters, and the component choices made during the design process, varies significantly depending on the domain it is used in. Some datacenter owners within the field of HPC may choose to focus on a hyperconverged infrastructure, where a many-core approach is taken to provide massive performance density. This allows for high performance, even within small metropolitan datacenters such as ones commonly found at universities, where space comes at a premium.

In a business-critical environment, hardware may be physically replicated across both racks and physical locations, so that even a total datacenter loss would have a reduced impact on business operations. In such a scenario, hardware is not chosen based on the suitability for one application, but to support the flexibility to be able to run any application in virtual machines. This is especially common in this field, with the intention of further reducing downtime. Virtual machines can be migrated between physical hosts while still running when hardware maintenance is required [14]. It is also possible to add resources to a virtual machine without power loss, dynamically scaling them to jobs.

Conversely, other datacenter owners may focus on scientific computing [30]. The requirements for such systems are different. They must support a high number of users, each with workloads that require as high a performance as possible. As a result, these datacenters are designed differently, with a high number of nodes running a baremetal operating system. The configuration of these nodes varies, depending on the intended use

of the node. Nodes are then categorized into performance tiers based on their intended use. Performance tiers are then chosen by customers depending on the kind of workload they want to run - for example, GPUs may be installed in the nodes in tiers intended for Machine Learning workloads.

Prefabs specific to certain domains can be represented in OpenDC, with different performance tiers included within each domain. In this way, users can easily build massive, homogenous systems by selecting the node specific to their performance target, and using it as a starting point.

3 Design of a Datacenter Hardware Representation

In this chapter, we address **RQ1**: *How to design a prefab abstraction that describes important technical and composable features of datacenter hardware to a high degree of fidelity?* To do this, we introduce our prefab concept, discuss how we envision it can be used, perform a requirements analysis on those use cases, and describe ways to work with prefabs to design datacenters.

Servers within a datacenter are typically feature rich, and often include specific features to differentiate them from their competitor’s. There are characteristics of the hardware (such as chassis colour) that have little relevance when designing a datacenter. Conversely, properties of the hardware such as the power draw, or the heat output, are important with regards to the planning of the datacenter infrastructure, as there will be a performance impact if these properties are not accounted for. When representing datacenter hardware in a data structure, it is important to consider which aspects of the hardware are relevant to represent.

3.1 Why Prefabs?

When designing our new data structure, it is important to consider the potential use-cases for key stakeholders. We identify our *stakeholders* to be researchers in the fields of distributed or large-scale computer systems, students in the same field, and datacenter operators in both industrial and academic settings.

Both researchers and students would benefit from the addition of prefabs to OpenDC in the same way that developers (and other customers) benefit from modern PaaS offerings (such as those from DigitalOcean), by focusing on which services they want to run in their datacenter without having to concern themselves with the underlying physical infrastructure. Additionally, datacenter operators using OpenDC for datacenter design and testing may also see a benefit from prefabs. Cloud service providers who provide *Infrastructure as a Service* (IaaS) services [24] typically use homogenous hardware configurations, with different configurations for each performance tier. With prefabs, it would be straightforward to create a prefab that is representative of a given performance tier, and then clone it when performing capacity planning during periods of growth. Lastly, given the widespread support for the sharing of information within academia, the ability to share prefabs would be an important addition to the features of OpenDC.

From these use-cases, we define five functional requirements for prefabs:

- FR1:** Users should be able to use existing topology definitions as a basis for new prefabs.
- FR2:** Users should be able to add prefabs into existing topologies, using them as building blocks to enable faster prototyping and design.
- FR3:** Users should be able to determine what tasks a given prefab is suited for, by means of a labelling mechanism. Prefabs could then be labelled to indicate, for example, that they are intended for highly parallelizable workloads.
- FR4:** Users should be able to easily share their prefab designs with others.

FR5: OpenDC should provide some example prefabs based on servers that are popular in the market. These prefabs would provide a building block for users to get started with.

These functional requirements cover the most fundamental tasks outlined in our potential use-cases by stakeholders. We choose these requirements as we believe that, together, they represent our core vision of the use of prefabs in OpenDC, enabling us to provide an answer to **RQ1** (*How to design a prefab abstraction that describes important technical and composable features of datacenter hardware to a high degree of fidelity?*).

FR1 and **FR2** are fundamental to our vision of prefabs. Without the ability to create and use prefabs as defined in these requirements, there would be no benefit to the use of prefabs, as they would effectively not exist. This presents the most significant challenge, as these two requirements dictate the basis functionality of our work. The implementation of these two requirements relies on the design, creation, and implementation of a data structure for prefabs into the existing architecture of OpenDC, as well as the design, creation and implementation of the interactions on prefabs defined in this requirements.

Conversely, **FR3** and **FR4** are not strictly necessary for the existence of prefabs, but are necessary for the sharing of prefabs. Several of the benefits of prefabs we have discussed come from the ability to contribute to, and benefit from, a wider body of work created by the community. As a result, we argue that these requirements, while not critical to the existence of prefabs, are necessary in order for prefabs to be useful to the broader community. Additionally, any benefit that prefabs may provide to making datacenter design more accessible exists in part through the ability for users to view and use prefabs that they did not have to design themselves. **FR4** is useful in this regard, as it allows for the ability to share designs. **FR3** serves then to support the functionality described in **FR4**, enabling users to more easily find prefabs that are of use to them. Therefore we choose to include these functional requirements focussed on the sharing and distribution of prefabs.

Lastly, we discuss **FR5**, which describes the inclusion of example prefabs in OpenDC. While this functionality is not necessary to the functionality of prefabs, we argue that it is necessary for the overall success of prefabs. Providing users with examples of existing hardware allows them to get started with OpenDC *out of the box*, supporting users in the period during which the wider library of community-created prefabs continues to grow. This functionality is particularly important, because OpenDC is an open source project. As a result, users can host their own instances of OpenDC, so the addition of example prefabs allows these users to benefit from prefabs in an environment where the wider community cannot contribute. It also is necessary for the adoption of prefabs by the community, giving users an understanding of what prefabs are. The challenge of implementing this requirement stems from there not being a validated method of identifying servers that are popular in the market. As a result, we must design and execute our own methodical approach to this problem.

A functional requirement that we consider, but ultimately do not include in this work, is the ability to edit prefabs. Although useful for users, we argue that the inclusion of this functionality is not fundamentally necessary to proving the concept of prefabs, and can therefore be explored in future work instead. We also consider the ability for users to choose the scope of their prefabs (e.g. a large prefab composed of a room, or a small prefab

composed of a single chassis). However, the technical complexity of this functionality falls outside of the scope of this work. Additionally, we choose to provide a consistent definition of what a prefab is in this work in order to make it clear to users what a prefab is, leaving the possibility for the exploration of this concept open to future work.

During the design process, we identify a further, more technical requirement; we define FR6 in Chapter 3.3.2.

3.2 Model for Datacenter Representation

In this work, we provide a model for representing datacenter hardware, allowing us to store information relevant to the simulation. Andreadis et al. created a similar model for datacenter hardware to model scheduling in OpenDC [2], which serves as a basis for the model presented here.

When designing the model, we find it important to consider that a large part of its purpose is to increase the usability of OpenDC. As a result, we choose to represent hardware characteristics that are possibly not used by the simulator, but provide useful information to the user. Our model, therefore, offers more detail by design than the one presented by Andreadis et al, storing a wider range of the characteristics of the hardware components within. For example, knowing the hardware brands is not necessary to simulate workloads, but such information is useful and often taken in to account when making design decisions or when presenting designs to a wider audience.

Our model considers different hardware components, each with different properties. These properties are detailed below.

3.2.1 Central Processing Unit(s)

A Central Processing Unit (CPU) in our model has four main characteristics: its name, its base clock speed in megahertz, the number of physical cores, and its energy draw in Watts. With the exception of the name, all of these properties are important when performing a simulation of the performance of the CPU.

A server is not limited in the number of CPUs it may contain. While commercially available servers, such as Lenovo’s ThinkSystem SR950², may contain up to 8 CPUs in a single chassis, we choose not to place an artificial limit on the CPU count. In this way, the design of the data structure remains extensible, for both future hardware, and for entirely theoretical simulations. Additionally, this provides the flexibility required to represent multi-chassis servers, such as HPE’s Superdome Flex³, that scale up to 32 sockets while functioning as one physical server to the operating system (and thus the workload).

3.2.2 Random Access Memory

In our model, Random Access Memory (RAM) has four primary characteristics. It has a human-readable name, a measure of its energy draw in Watts, its capacity in megabytes, and its bandwidth in megabytes per second.

The bandwidth of the memory is the product of its clock frequency, the number of data transfers per clock, the bus width of the memory, and the number of interfaces (modules)

²<https://lenovopress.com/lp0647-thinksystem-sr950-server-xeon-sp-gen-1>

³<http://hpe.com/superdome>

within the system. At the time of writing, server memory conventionally uses Double Data Rate (DDR), with a 64-bit bus width. The bandwidth of each module is calculated as if it were the only module in the system. The combined memory bandwidth of a system therefore increases linearly as we add modules.

We choose again not to limit the number of memory modules that can be added to a system. While our findings suggest that Lenovo’s aforementioned ThinkSystem SR950 has the most memory modules of any single-node system, with 96 in total, we do not want to artificially constrain researchers and datacenter operators from experimenting with theoretical systems with much higher memory density.

3.2.3 Graphics Processing Unit(s)

When representing Graphics Processing Units (GPUs), we represent its name, the number of GPU cores it contains, the clock speed of the GPU cores, and the power draw.

For the purposes of supporting theoretical hardware definitions, there is no limit on the number of GPUs that can be added to a topology. This choice additionally supports the simulation of systems that use an external chassis to house GPUs, using PCI Express to connect to the main compute chassis, such as the One Stop Systems 3U Compute Accelerator⁴, which supports up to 16 GPUs per external chassis.

3.2.4 Storage

In our storage representation, we model the properties currently required for storage simulation within OpenDC. These include the name of the device, its size in megabytes, its peak read speed in megabytes per second, and its peak power draw in watts.

There is no limit on the number of drives that a system can include, in an effort to support theoretical hardware definitions. Additionally, this lack of limit allows for the simulation of systems that may rely on external disk arrays (such as NetApp’s DS460C⁵) for storage, as OpenDC currently does not include support for storage topologies such as *Storage Area Networks* (SANs).

3.3 Definition and Design of a Data Structure for Storing Prefabs

3.3.1 Definition of a Data Structure for Storing Prefabs

The addition of prefabs to OpenDC will enable users export their datacenter topologies in a manner that allows them to be easily reused. These prefabs contain a copy of the topology, including all the hardware represented within the chosen topology, such as CPUs, GPUs, memory, and storage. A visual depiction of the structure of our prefab definition is given in Figure 4, by means of a UML class diagram. As shown, each of these units of hardware retain any detailed specifications contained in the original topology object, such that it forms an exact copy of the source topology.

In addition to these hardware components, prefabs also contain certain metadata. This metadata includes information such as the prefabs date of creation, and the date of its last

⁴<https://www.onestopsystems.com/3u-compute-accelerator-nvidia-tesla-gpus>

⁵<https://www.netapp.com/us/products/storage-systems/disk-shelves-and-storage-media/index.aspx>

modification. It also contains information that supports some of our functional requirements, in particular **FR3** and **FR4**. We include an array containing user-created tags, for prefab classification, as well as the author’s user ID, so that work on a given prefab can be accredited to an individual. We also include a field in the prefab data structure indicating the visibility of a prefab, determining whether it is searchable at all.

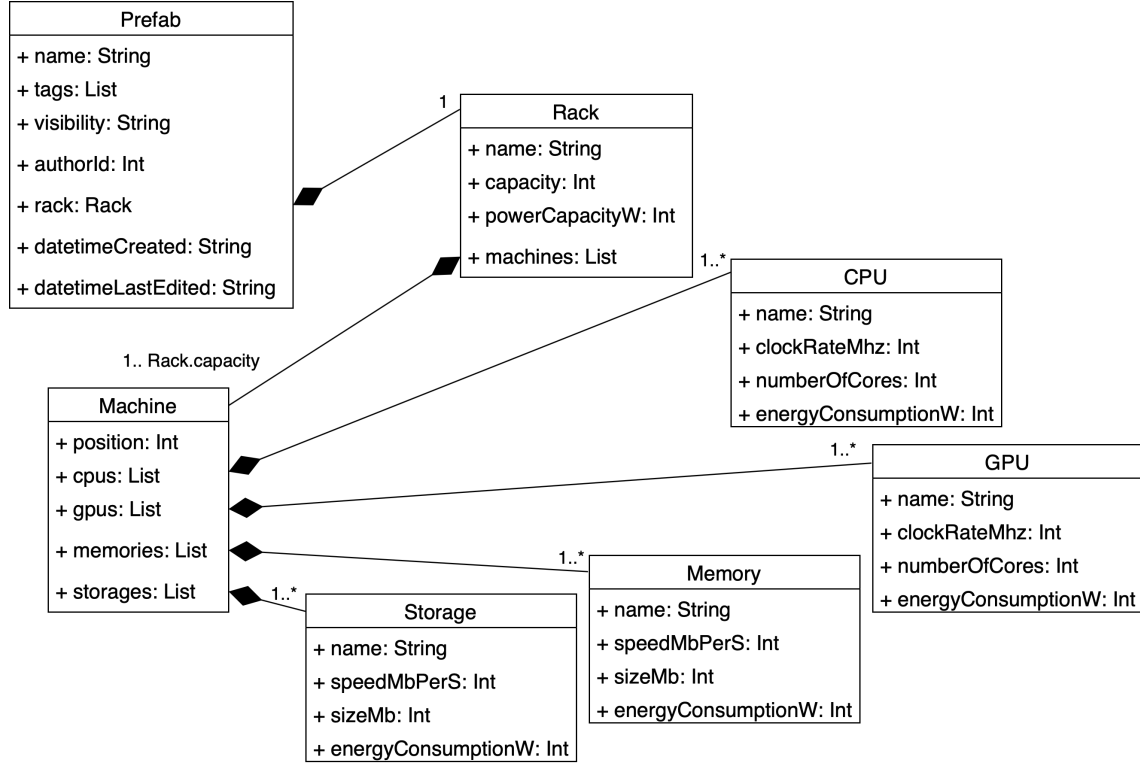


Figure 4: A UML class diagram depicting the definition of the prefab data structure.

3.3.2 Design of a Data Structure for Storing Prefabs

To store our datacenter hardware representation within OpenDC, we design a new data structure. In order to do this, we must first select a new format for storing the data structure. We do this based on the features available in each format, as well as the ease of implementation of the format. Next, we review the definitions put forward in Chapters 3.2 and 3.3.1, and determine whether there is any additional data that might be useful to represent. This helps us ensure that our design has a sufficient degree of fidelity. Lastly, we create a data structure in our chosen format that represents the chosen characteristics, representing our chosen features in the chosen format. This forms our data structure.

When considering a new format for representing our data structure, we must first identify a suitable format that meets the another requirement necessary for this project:

FR6: The storage format must be human-readable, and compatible with the existing OpenDC software stack.

Readability of the data structure allows for easier extensions in the future by making the data structure easier to debug, recreate, and test with. This is especially useful

when working with extensions that interact with the API. The readability of the format also supports possible future work, that may extend OpenDC to include a console for programming prefabs and topologies.

The database used in version 1.x of OpenDC contains 35 SQL tables, 20 of which are used to store topologies. As a result, adding hardware items to the database requires a complex set of queries, with deletions and modifications requiring additional queries. To reduce the complexity associated with OpenDC, we seek a storage format that allows us to store entire prefabs (and topologies) within a single object. The advantage of such an approach is that the database then supports the adding, updating and deleting of prefabs and other topology objects, without the use of complex queries. We therefore consider several new formats for representing our data structure; after careful consideration, we select the *JavaScript Object notation* (JSON) format. We now describe the process of selection.

First, we consider *YAML Ain't Markup Language* (YAML)⁶ as our document format. YAML supports nested object hierarchies, however we argue that YAML is not suitably human-readable. It relies on indentation for distinguishing levels of hierarchy, with no clear boundaries delimiting separate objects.

We also consider *Extensible Markup Language* (XML)⁷, which is a common language for storing and representing documents that is used in many web applications. It is both human and machine readable by design, and is well suited to nested object hierarchies, making it a strong contender to be used to store prefabs. However, XML requires parsing. The current OpenDC frontend is written in **ReactJS**, which does not natively support parsing XML without the use of an external library. As we prefer not to add additional dependencies to OpenDC, we do not find XML to be a suitable choice as a document format.

Lastly, we consider *JavaScript Object notation* (JSON)⁸. JSON is an expressive, industry-standard object storage format that is widely used in web-based applications. It is both human-readable, and supports nested object hierarchies. The support for nested objects is crucial to us to achieve some of our functional requirements, namely **FR1** and **FR2**, as it allows for easier insertion into the object. The human-readability of JSON aligns with **FR6**, making it easier for contributors to extend the data structure in the future. In addition, it is supported by the standard libraries available in both Python and ReactJS, making it ideal for our data structure storage format. As a result, we choose JSON as a means to store our prefab data structure.

Our JSON-based data structure aims to be simple to understand, and easily expandable to represent hardware configurations that we do not focus on in this research (i.e. blade servers, or other chassis that may contain multiple/unconventional motherboards). This means that, along with the default fields for components we have previously specified, it is easy to add new fields to components in the future by simply adding new fields to the object.

Once we have decided on our storage format, we can consider the data we wish to represent. In Chapter 3.2, we discuss the individual components used in datacenters, and

⁶<https://yaml.org/spec/1.2/spec.html>

⁷<http://www.w3.org/TR/REC-xml>

⁸<https://www.json.org/json-en.html>

which characteristics of these components are important to represent. However, these components cannot exist in a datacenter outside of a chassis, which also must be stored within a rack. As such, we define a hierarchical relationship between subcomponents and the chassis, and between the chassis and the rack. As a result, we define important characteristics for each of these objects.

A Rack holds one or more chassis, as well as networking, power, and storage hardware (such as drive enclosures). In our topology representation, we only consider the chassis being stored within a rack, as OpenDC currently does not represent networking, power, or storage hardware. The amount of hardware that a rack can contain depends on the size of the rack, measured in *Rack Units* (U), with industry-standard full-height racks having room for 42U, and half-height racks having capacities between 18 and 22U [11]. We argue that, for the purposes of datacenter planning, it is important to characterise the capacity of server racks within OpenDC. As a result, we include a capacity property in the rack object, as well as a list containing each machine object stored within the rack.

All of the machines in a rack require power to run, with racks often being provisioned for a certain power budget. As a result, we should also represent this power limit, in order to support the planning of power infrastructure. We choose to do this by adding a property to the rack object representing its power capacity in Watts.

Lastly, in order to be able to visually differentiate racks within OpenDC, we include a property to store the name of a rack as a string.

We represent chassis in the topology by means of a Machine object. This object contains four lists, one for each type of subcomponent (e.g. CPUs, GPUs, memory, and storage). Additionally, it contains a property indicating its position in the rack, by means of an integer. By means of this position value, we can order the way machines are displayed in the OpenDC frontend.

Lastly, we include the parent of the Rack object: the Prefab object. In Chapter 3.3.1, we discuss the inclusion of tags, an author property, and visibility to assist with sharing. Prefabs also contain a name, the date and time of creation, and the date and time of the last edit made to the prefab.

```
{
  "name": "Intel Xeon Gold 6252",
  "clockRateMhz": 2100,
  "numberOfCores": 24,
  "energyConsumptionW": 150
}
```

Figure 5: A CPU represented in our JSON data structure format.

Now that both the storage format, as well as the design of the data structure have been decided upon, we can create our data structure in our storage format. In order to create the prefab object in JSON, we begin with the individual components at the lowest level of the hierarchy: the CPU, GPU, memory and storage. Each of these components must be instantiated as a JSON object. An CPU in this format is given as an example in Figure 5. Once each component has been created, they can be added as nested children of

their parent objects. For example, a CPU object is added to the `cpus` list in the Machine object, which is in turn added to the `machines` list in the Rack object. Lastly, the Rack object is added as the `rack` value in the Prefab object. An abbreviated version of a prefab created according to this method is shown in Figure 10.

3.4 Designing Ways to Create and Interact with Prefabs

When determining interactions with prefabs, we must define a set of operations that meets the functional requirements previously defined.

From **FR1**, we can make the case for functionality that would allow a prefab to be created from an existing topology. Such functionality should create a copy of the current topology, and store it as a prefab, allowing the user to determine the visibility (private or public) and name of the prefab.

The ability to save topologies as prefabs also serves as the basis for the behaviour highlighted in **FR2**, where users would be able to then create new topologies from prefabs, or add prefabs into their existing topologies. Users would be able to choose from their own private prefabs, and from all public prefabs, when adding prefabs into their design.

Users would also be able to find prefabs based on their intended use, amongst other labels. Such behaviour would be made possible by means of a labelling mechanism, satisfying **FR3**. This labelling mechanism would take the form of an additional field on the prefab object, storing an array of tags assigned to the prefab. These tags would indicate the suitability of the prefab to a specific task, but could also be used to indicate the scale at which it can execute such a task. Such labellings are already used to convey the size of IaaS instances offered by cloud providers [10], and could also be used by OpenDC to convey the size of workload a prefab is designed to execute.

The aforementioned labelling mechanism, combined with the ability to mark a prefab as public or private, would be crucial to enable the sharing of prefabs between users mentioned in **FR2**. Such a sharing mechanism should allow users to search for prefabs, or filter them based on common tags. This would enable users to find prefabs created for specific tasks, and easily share prefabs for new tasks with the community, as highlighted in **FR4**.

Lastly, the inclusion of some OpenDC prefabs outlined in **FR5** is important to provide a basis for new prefabs. Such a provision would also provide a reference for how prefabs can look to users unfamiliar with them. It would also provide the means to immediately get started with creating topologies from prefabs, instead of waiting for the community to begin providing their own. Additionally, the inclusion of some basic prefabs in OpenDC would allow users to benefit from prefabs even if they are self-hosting OpenDC, without the community contributions.

3.5 Applications of Prefabs

Based on our design for prefabs, we envision that prefabs have a wide variety of potential uses.

One example of such a use would be to capture datacenter features of domains of societal interest, making them available as a library of prefabs for users of OpenDC. This would provide a starting point for users intending to explore a wide variety of domains, making

datacenter design in OpenDC accessible to a wider audience of users. In Chapter 5.1, we present an approach to capturing datacenter features for a specific domain (namely High Performance Computing).

Another important potential application of prefabs would be to enable an increase in the speed of datacenter design, while retaining the same level of detail. Currently, designing a topology requires lots of interactions by the user, as they must build the entirety of every design from the ground up. However, with prefabs, it would be possible to save topologies to be reused later, and load prefabs into topologies, reducing the time taken to build designs that rely on common components.

Prefabs also provide utility when designing state-of-the-art datacenters for use in simulation-based experiments. Prefabs can be labelled according to their intended workload, improving experiment reproducibility through the sharing of prefabs. Research could include prefabs as artefacts of the work. This would make it trivial for others in the field to include these prefabs in their own topologies when replicating, or even furthering, the research of others. Such an approach to experiments is already being utilised by others within the OpenDC team.

When teaching popular datacenter concepts, prefabs could also serve a purpose. By creating prefabs of famous (or otherwise historic) datacenter or supercomputer designs, students could use OpenDC to explore characteristics of these examples. Students would also be able to explore the consequences that adding or removing certain hardware (such as GPUs) has on a workload. Lastly, teachers could use OpenDC to provide demonstrations of how datacenters are designed, organised, and built, providing a valuable learning experience.

Lastly, prefabs could be used for the presentation of datacenter design ideas. A future application of OpenDC could even be during discussions regarding the design of *The Distributed ASCI Supercomputer 6* (DAS-6)⁹, the successor to the DAS-5 [3]. Prefabs within OpenDC could be used as part of the design process, with prefabs being used to store the homogenous nodes that will likely comprise the cluster. Then, designs using these prefabs could be presented to the project partners, with simulations run on these topologies used to show the efficacy of one design over another.

⁹<https://www.cs.vu.nl/das/>

```
./prefab.py
OpenDC Prefab CLI
> help
Usage: prefab add <prefab>:      imports a prefab from JSON
      list:                      lists all (public) prefabs
      export <prefab> [json|yaml]: exports the specified prefab to t
he specified filetype (with JSON used by default)
      clone <prefab> [new prefab name]: clones the specified prefab,
giving the new prefab a name if specified
      remove <prefab>:          removes the specified prefab from the databas
e
> _
```

Figure 6: Prototype showing the CLI running the `help` command. The output lists the main operations it supports.

4 Implementation of a Prototype

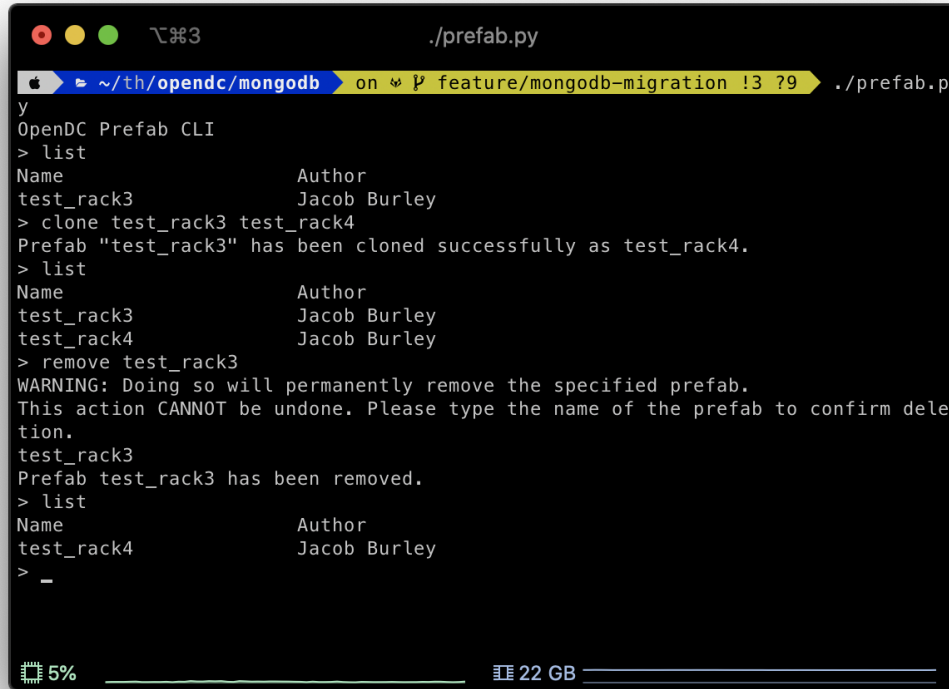
In this chapter, we address **RQ2**: *How to implement a prototype of the prefab abstraction?* To do this, we explain our reasoning behind the use of a prototype, discuss the development of a prototype, and detail the varying lessons learned during this process.

4.1 Why Prototype?

Prototyping is an important part of our design process. To further the goal of simplifying working with OpenDC, we have chosen to transition the storage database away from SQL, towards a NoSQL design.¹⁰ MongoDB has been chosen by the OpenDC team as a NoSQL document storage solution due to its ease of use, and its flexibility. MongoDB additionally natively supports the insertion of JavaScript Object Notation (JSON) objects, which we use as the storage format for our data structure. It also provides strong library support for Python via the `pymongo` module, supporting the integration of the new database structure with a new implementation of the OpenDC API.

In version 1.0, OpenDC uses a combination of ten different technologies to provide its service. In the frontend, JavaScript, React, Redux, and Konva are used. For the webserver and API, OpenDC uses Python, Flask, FlaskSocketIO, and OpenAPI. MariaDB

¹⁰The author of this thesis was involved in the transition from SQL to NoSQL, involving the implementation of the new database, and the porting of the current database schema to MongoDB. This effort also involved porting the existing OpenDC API from Python 2.7 to Python 3, and rewriting the API to include new endpoints for prefabs, and the new topology structure.



```
./prefab.py
~/th/opendc/mongodb on feature/mongodb-migration !3 ?9 ./prefab.p
y
OpenDC Prefab CLI
> list
Name          Author
test_rack3     Jacob Burley
> clone test_rack3 test_rack4
Prefab "test_rack3" has been cloned successfully as test_rack4.
> list
Name          Author
test_rack3     Jacob Burley
test_rack4     Jacob Burley
> remove test_rack3
WARNING: Doing so will permanently remove the specified prefab.
This action CANNOT be undone. Please type the name of the prefab to confirm deletion.
test_rack3
Prefab test_rack3 has been removed.
> list
Name          Author
test_rack4     Jacob Burley
> -
```

Figure 7: Prototype demonstrating the import and deletion of a topology module.

is currently used as the database for storing topologies, simulations and experiments, and the simulator itself is written in Kotlin. Due to the significant change to the software stack that a transition to MongoDB represents (represented graphically in Figure 8), we must use our prototype to test the compatibility of our design in the existing context of OpenDC to make sure that our design (and MongoDB) is a viable addition to OpenDC.

As a result, prototyping provides us with a way of becoming familiar with these new technologies, and assessing their suitability, before we begin the process of implementing them into the existing OpenDC codebase. We also can create prototypes of the data structure, and the corresponding interactions with it, and assess the suitability of the data structure with regards to the interactions we require it to perform.

While it would be possible to continue with the implementation of our design without any prototyping whatsoever, such an approach would severely impact our development time. Because the codebase of OpenDC is large, compiling it in full for minor changes in order to test functionality would substantially increase the duration of each development iteration. The value created by prototyping is through allowing for quick, small-scale, and discardable test implementations that we can learn from at a fast pace. By extending the time taken to create a prototype, the sunk cost with regards to time becomes greater for each prototype, potentially prohibiting us from discarding ideas that do not add sufficient

progress towards our goals. This would also impact the testing of new technologies, as even small changes in our configuration (e.g., for MongoDB) would cost us more time. As a result, the quality of our implementation may be harmed as we retain code that isn't worth the perceived effort to improve.

4.1.1 Requirements Analysis for a Prototype

When creating our prototype, it is important to keep the goals of this process in mind. As previously discussed, we want to explore and become familiar with MongoDB, and determine which changes need to be made to the existing OpenDC codebase to support the addition of this new technology. This will involve the creation of a Python backend to interact with the database, allowing us to provide our own standardized methods of performing common tasks throughout the codebase of OpenDC. Additionally, we want to iteratively improve the design of our data structure using our prototype. Our prototype should thus be lightweight enough to support quick experiments using the data structure, allowing us to engage in iterative design and improvement of our data structure. Lastly, a prototype should represent a minimal subset of the concepts found in OpenDC. With MongoDB providing for the database, with a Python backend implemented to standardize our desired database operations, we require a frontend through which we can interact with the prefabs system. Through this frontend, we should be able to perform operations on the prefab data structure we have designed, allowing us to carry out tests to determine whether our operations and data structure function according to the functional requirements defined in Chapter 3.1.

As such, we can define three implementation requirements for our prototype.

- IR1:** The prototype should use MongoDB for its database implementation.
- IR2:** The prototype should include a frontend implementation, through which interactions with prefabs can be tested.
- IR3:** The prototype should support iterative development of the data structure, through being lightweight enough for rapid experimentation.

4.2 Prototype Development

Our three main goals in prototyping are to explore how we can best implement and interact with MongoDB (as mentioned in **IR1**), to test our prefab data structure and interactions through a frontend (**IR2**), and to iteratively improve our data structure design (**IR3**). As a result, the prototype consists of three components: a MongoDB instance, a Python module that interfaces with the database, and a second Python application to serve as the frontend. This configuration has been chosen to be relatively close to how OpenDC already implements its database connections, which also uses a Python module to abstract away most database interactions.

MongoDB is chosen as a database replacement, as it easily facilitates the storage of our JSON objects. MongoDB stores documents in a JSON-like format internally, and has strong library support for inserting, modifying and exporting said documents, making it relatively straightforward to re-implement our database connection.

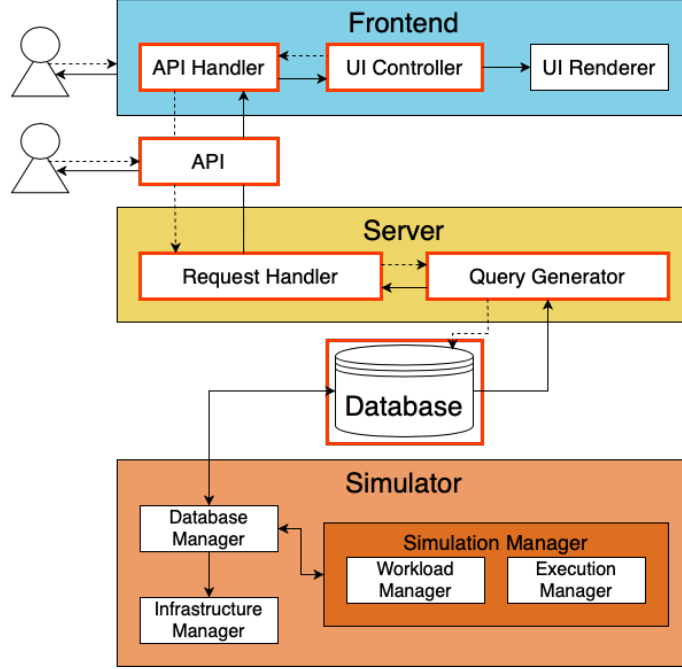


Figure 8: A Component Diagram of OpenDC. Components modified in this work are highlighted.

Because we are able to insert nested JSON objects into MongoDB, we do not need to rely on multiple tables and foreign keys used in a typical relational database like MariaDB. We can simply store the entirety of a topology in a single document within MongoDB, and access specific fields within those documents when we need to. As a result, our implementation of MongoDB relies on simpler queries than SQL, where each component of a rack had to be defined with one query, requiring another query for each insertion of said component into a server chassis. Adding a full rack to the MongoDB database requires only a single query to achieve the same effect, regardless of the number of machines. The simplicity of this design, enabled by the use of nested JSON objects, also means that we do not have to update each table and foreign key when we modify a prefab, as we can simply fetch the whole object into memory, modify it, and re-insert it into the database instead.

For our frontend, we opt for a simple *Command-Line Interface* (CLI) instead of a *Graphical User Interface* (GUI). We make this choice as we feel that the addition of a GUI would be time-consuming, and over-complicate the prototype. This would defeat the purpose of the prototype, which is to be simple. The resulting prototype frontend takes the form of a CLI for interacting with prefab objects (shown in Figure 6). This prototype is capable of performing some basic operations on prefabs, such as importing a prefab from a JSON file on the local filesystem and adding it to the database, exporting a prefab to a JSON file on the filesystem, cloning a prefab, listing all prefabs available in the database, and deleting prefabs. It even supports a basic notion of authorship, whereby listing all public prefabs also displays their authors. Although the prototype frontend lacks an authentication mechanism into which this functionality could be fully integrated,

it neatly presents a concept important to our design. The behaviour described can be seen in Figure 7.

The prototype is also designed to enable rapid testing and experimentation. To this end, we extend the the frontend so that it can be run in one of two ways: either as an interactive shell, or as a single command that takes arguments. This latter method of execution allows for batch scripting of tests during the prototyping process, enabling us to iterate even faster when testing features of the data structure.

The functionality modelled covers the most essential functions that a prefab implementation requires, with the listing functionality even including some rudimentary access control support. It also serves as a useful test of the functionality of MongoDB, meeting **IR1**, and provides sufficient reference material for the implementation of a MongoDB connection in the API of OpenDC. The prototype also includes a working frontend, allowing for interactions on prefabs, as required by **IR2**. Lastly, through the lightweight structure of the prototype as well as some additional functionality in the frontend, our prototype supports the goal of iterative development defined in **IR3**.

4.3 Lessons Learned

During prototyping, we learnt many things that influenced the decisions we made during the implementation phase.

Firstly, we opted not to enforce schema validation on the collections within our database. The reasoning behind this decision was that schema validation in the database added unnecessary complexity, and limited the shape of our data (and thus hardware that could be represented). We instead choose to check for required fields (such as the topology name, or the prefab visibility) within the API. This allows for topologies to contain hardware configurations that may not have been considered during the design process, while also allowing us to make assumptions about certain aspects of the data structure.

We also became more familiar with certain aspects of MongoDB, including the differences between Binary JSON (BSON) and JSON. When we began working with MongoDB, we operated under the assumption that MongoDB would export objects in JSON, just as it can be given JSON objects to import. However, this assumption turned out to be incorrect. During testing of topology import functionality, we attempted to import a topology that we had exported to JSON. When this test failed, we discovered that MongoDB does not use pure JSON for exports. Instead, it exports objects in BSON, the same format it uses to store objects internally within the database. MongoDB simply stores files in a binary format internally, and converts them back to something human-readable when you export them. However, the human-readable format that MongoDB uses is not JSON, but is deceptively similar: BSON uses single quotes (') where JSON uses double quotes(""). As a result, we implemented a conversion from BSON to JSON within our API, so API responses are formatted in proper JSON that can be used by the frontend.

4.4 Future of the Prototype

The prototype detailed here will be released in the *Free and Open Source Software* (FOSS) repository for the OpenDC project¹¹. Additionally, it is scheduled to be released in the

¹¹<https://github.com/atlarge-research/opendc>

next official release of OpenDC, where it will be available for use in supporting further extensions to the OpenDC project.

Operator	Time taken (seconds)
K (keystroke or button press)	.75
P (point to a target on a display with a mouse)	1.1
H (homing hands on keyboard or other device)	0.4
M (mentally preparing for executing physical actions)	1.35
B (mouse button press or release)	0.1
R (system response time)	0.1

Table 1: Relevant KLM operators used in our testing, and their times to complete [28].

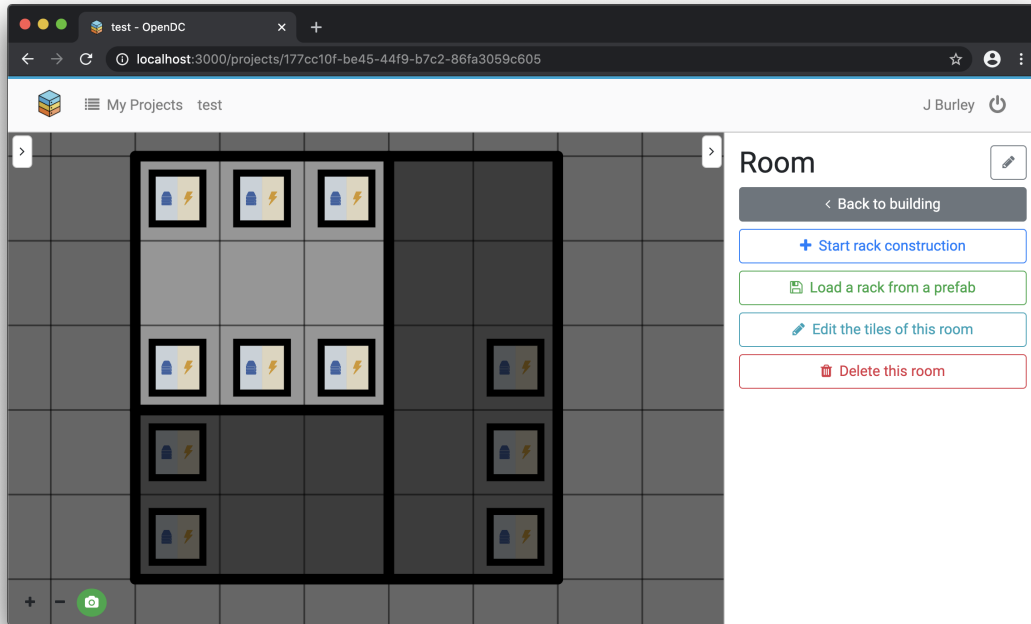


Figure 9: The starting position for evaluation. Screenshot taken from OpenDC extended with our work.

5 Evaluation of Design & Implementation

In this chapter, we evaluate the success of both the design, and the implementation of our prefab data structure. We do this in two ways: first, we test the ability of prefabs to represent real-life hardware, addressing a part of **RQ3** and as per **RQ1**. Secondly, we use a popular Human-Computer Interaction model to assess the success of the implemented operations on prefabs, thus addressing the second part of **RQ3**.

Company	Units Shipped	Market Share
Dell Technologies	474,011	18.4%
HPE/New H3C Group	377,544	14.7%
Inspur/Inspur Power Systems	211,007	8.2%
Lenovo	153,570	6.0%
Super Micro	132,001	5.1%
ODM Direct	770,446	29.9%
Rest of Market	456,841	17.7%
Total	2,575,439	100.0%

Table 2: Market share distribution amongst server OEMs in Q1 of 2020.

Brand	Model	CPUs	DIMMs	Drives	GPUs
Dell	PowerEdge R440	2	16	10	0
Dell	PowerEdge R640	2	24	10	0
Dell	PowerEdge R740xd	2	24	24	0
Dell	PowerEdge R940	4	48	24	0
HPE	Superdome Flex Node	4	48	4	4
HPE	DL360 gen10	2	24	8	0
HPE	DL380 gen10	2	24	24	0
HPE	DL580 gen10	4	48	48	0
Inspur	NF5280M5	2	24	25	0
Inspur Power Systems	FB5180G2	2	16	10	0

Table 3: Overview of the HPC prefabs created for this thesis.

5.1 Evaluation of the Design with Domain-Specific Prefabs

In this section, we evaluate our prefab data structure design, demonstrating that it can support the design of today’s state-of-the-art datacenters. We are specifically interested in datacenters operating in societally important domains, and evaluating that our work can support designs using professional equipment popular in the market recently. Concretely, we explore the High Performance Computing domain, and validate our prefab representation by creating prefabs for some of the HPC-oriented server offerings from the top 3 OEMs by market share in Q1 2020.

5.1.1 Representing Domain-Specific Hardware with Prefabs

To select server models to represent, we identify the top server OEMs by market share in Q1 of 2020 [25]. These summarised findings are presented in Table 2. We choose the three individual manufacturers with the largest market shares (highlighted in green) as our target manufacturers¹². We then choose the HPC-oriented server offerings from each of these manufacturers, and attempt to model them in the prefab data structure.

The server selection process varies slightly by manufacturer, but generally, we seek out servers that the manufacturer describes to be suited for HPC, or other intensive compute workloads. For each server, we then attempt to represent it with as powerful of a configuration as possible, populating all sockets, DIMM slots and storage bays. For HPE’s Superdome Flex, we also include the optional GPUs in the configuration. Other configurations present support GPUs, but as they are not included in any of the available OEM configurations, we opt not to include them in these configurations. Additionally, we choose not to include hardware from New H3C Group, as these servers are rebranded variants of the hardware already produced by HPE. As a result, the configurable options remain exactly the same as those of their HPE counterparts.

The variety of hardware configurations in the resulting prefabs is shown in Table 3. In this table, we show the brand and model of each server chosen. We also represent the configuration of each server, in terms of the number of hardware components included. The types of components shown are the components that our prefab representation supports.

From this table, we can see that several of the selected server models offer very similar configurations in terms of hardware. In fact, all but two of the models we recreated offer 12 *Dual In-line Memory Modules* (DIMMs) per CPU socket. As a result, the only characteristics by which some of these prefabs are differentiated by are the number of GPUs included, or the number of storage drives it can support. During design of a datacenter, designers may have to consider the different models of server offered by different brands. While many of the above designs are remarkably similar in terms of hardware, they may exhibit different performance characteristics, especially under different workloads. By providing prefabs such as these in OpenDC, users involved with datacenter design can compare each model in terms of performance at the target workloads, something that would otherwise be prohibitively expensive. As a result, designs tested in this way would become more performant, leading to improvements in datacenter efficiency.

In Figure 10, a truncated JSON representation of a server model (specifically, a Dell PowerEdge R440) is given. This representation includes the metadata surrounding the prefab (as discussed in Chapter 3.3.1), as well as a depiction of the hierarchical layout described in 3.3.2. This hierarchical layout provides a logical structure to the prefab, and organizes components where they can easily be found. In addition, it expresses the relationships between the components and the chassis: components can’t be used in a datacenter scenario without being enclosed in some kind of chassis. As a result, this design is clearly readable and understandable, conveying its contents quite clearly. This prefab represents all components necessary for simulation of this server, matching the detail provided on the product page for this server¹³. It does, however, lack speciality components, such

¹²There was not sufficient information surrounding the *Original Design Manufacturers* (ODMs) included in the "ODM Direct" grouping to determine whether any individual ODM would fall into our selection on the basis of their direct sales to customers. As a result, we choose to exclude ODM manufacturers.

¹³<https://www.dell.com/gd/business/p/poweredge-r440/pd>


```

{
  "_id" : 440,
  "name" : "Dell R440",
  "tags" : ["hpc"],
  "visibility" : "public",
  "authorId" : "78532789067890245",
  "rack" : {
    "name": "Dell R440",
    "capacity": "42",
    "powerCapacityW": "25000",
    "machines" : [
      {
        "position": 1,
        "cpus": [
          {
            "name": "Intel Xeon Gold 6252",
            "clockRateMhz": 2100,
            "numberOfCores": 24,
            "energyConsumptionW": 150
          },
          ...
        ],
        "gpus": [],
        "memories": [
          {
            "name": "SK Hynix RDIMM HMA84GR7MFR4N-VK",
            "speedMbPerS": 42656,
            "sizeMb": 32768,
            "energyConsumptionW": 10
          },
          ...
        ],
        "storages": [
          {
            "name": "Dell 1.92TB SSD SATA",
            "speedMbPerS": 600,
            "sizeMb": 1920000,
            "energyConsumptionW": 10
          },
          ...
        ],
      }
    ]
  }
}

```

Figure 10: An example of a prefab for a Dell PowerEdge R440.

as Dell’s *Integrated Dell Remote Access Controller* (iDRAC)¹⁴, used for remote lights-out management [5]. While these components are not necessary for simulation, it is important to consider that the presence of these components may be crucial in influencing purchasing decisions. As such, it may be worth further consideration to include representations of components such as these for the purpose of presenting designs.

5.1.2 Fidelity of Hardware Representation

Ultimately, we argue that the fidelity of our hardware representations (a condensed example of which is given in Figure 10) is sufficient. The characteristics of the chosen hardware representable in our data structure are sufficient for simulation of that hardware in OpenDC, so the prefabs created are functional in terms of being able to use them as part of a topology to be simulated. However, there are aspects of each server that are not represented in our prefabs, such as the presence of Out-of-Band management, or network cards. These components currently do not form part of our topology design, as they are currently not relevant to our simulations.

Additionally, there are aspects of hardware that we have chosen not to represent in this thesis, such as multi-node chassis. As a result, it was not possible to model hardware such as HPE’s “Apollo” line of high-density HPC servers¹⁵, or Dell’s PowerEdge Blade Enclosure¹⁶. Such hardware is seen frequently in the HPC domain, providing high levels of compute density, further solidifying the notion that OpenDC should support these kinds of representations in the future. However, it is still possible to simulate performance aspects of these multi-node systems, as they can be represented within OpenDC as individual servers. As a result, this limitation only impacts physical datacenter design.

Lastly, the “Inspur Power Systems FP5180G2” prefab (based on the homonym entry in Table 3) contains IBM POWER9 processors, based on the Power Instruction Set Architecture [16]. These CPUs exhibit different performance characteristics than Intel or AMD processors, in part due to different instructions exposed by the CPU. This difference is not accounted for by the simulator in OpenDC, and there is currently no provision for representing this difference in our topology structure. This is an improvement that could be made in the future, and would be relatively straightforward to represent given the flexible nature of our topology structure.

5.1.3 Applying the Prefabs Approach to Other Domains

We argue that this approach would be straightforward to apply to other domains. Most manufacturers split their hardware offerings up according to their intended workloads, allowing for relatively straightforward server selection. For smaller domains (such as game streaming), however, it may be beneficial to consider the performance requirements of the domain with regards to specialized hardware, as these domains are not always as clearly differentiated.

¹⁴<https://www.delltechnologies.com/en-us/solutions/openmanage/idrac.htm>

¹⁵<https://www.hpe.com/us/en/storage/apollo-4000.html>

¹⁶<https://www.dell.com/ly/business/p/poweredge-m1000e/pd>

5.2 Evaluation of Prefab use in Practice

In this section, we evaluate the performance improvements resulting from the use of prefabs in practice. We use the *Keystroke-Level Model* (KLM) [28] to test the duration of performing a given task in both versions of OpenDC, and compare the results. KLM measures the number of keystrokes (or other actions) that need to be executed to complete a given goal. Each action is assigned a time duration. To calculate an estimate of how long a task will take, one can map out the actions required to complete the task and translate these actions directly to a time duration.

5.2.1 Experimental Setup

When evaluating our implementation, we must compare our additions to OpenDC to a version of OpenDC before our extensions were added. For the purposes of our comparison, we will use the aforementioned KLM methodology to test the duration of executing a specific task. We present a subset of the KLM operators in Table 2.

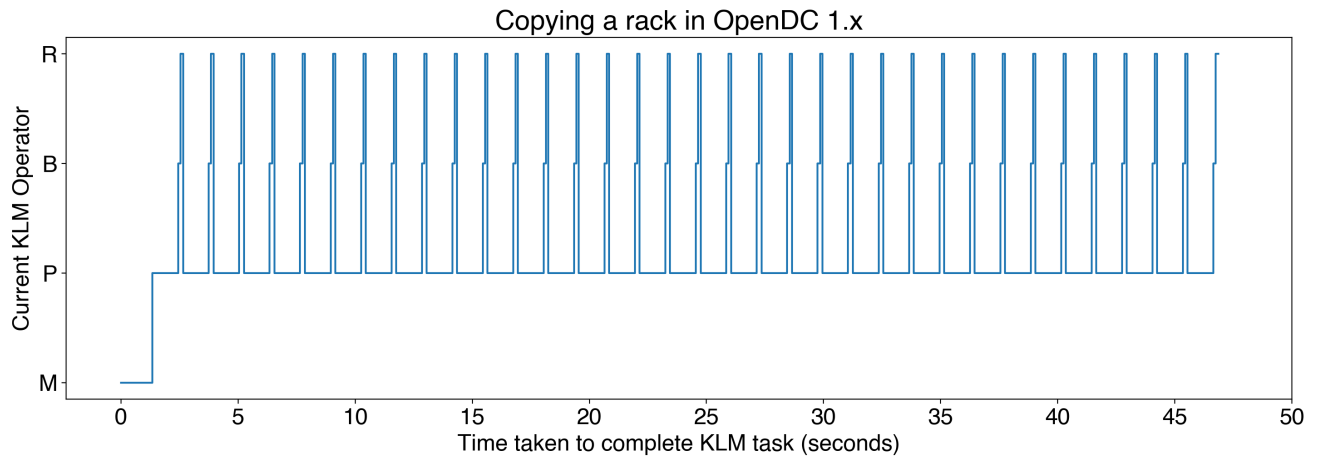
As part of our testing, we account for the response time of the system. While we observed that, once the application is loaded locally, it reacts quite quickly (almost instantly) to user interaction. However, in this scenario the application is running on a local machine, with only one user. In a production environment we would expect both multiple concurrent socket connections due to multiple users, and increased latency between the server and the client when they are not on the same host. As a result, we use an R value of 0.1 seconds to represent the time the system takes to update the user interface, as we feel this is more representative of how the system would behave when deployed. This value will be added to our final total for each click of the mouse (B). We also make the assumption that the user has their hand on the mouse already, negating the need to include the duration required to home their hand to the mouse.

For our usability test, we compare the process of copying a rack in version 1.x of OpenDC, and our new version of OpenDC with prefabs. We create a room, containing a single rack. This rack contains 3 machines, each containing 1 CPU, 1 GPU, 1 memory DIMM, and one hard disk. We then attempt to copy the contents of this rack into a new rack. The KLM task begins on the screen shown in Figure 9, and ends when a second rack has been created with the same contents as the initial rack.

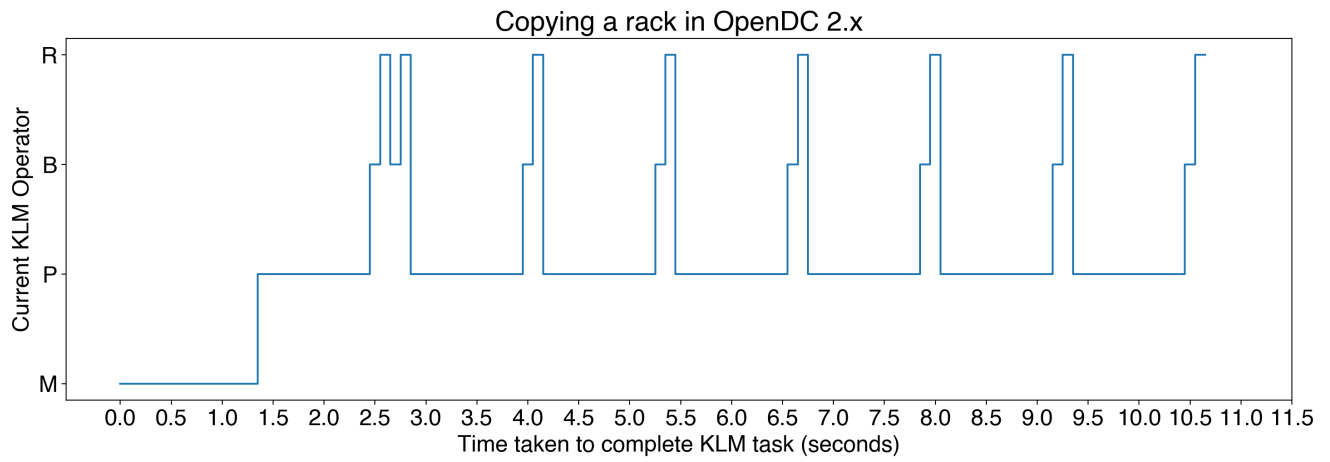
5.2.2 Experimental Results

From the results presented in Figures 11a, 11b and 12, we can draw two conclusions.

The first of these is that the number of interactions with the system is significantly reduced with the addition of prefabs. In Figure 11a, it can be seen that the process of copying the rack takes 70 discrete interactions. With less machines in the rack, version 1.x requires fewer interactions to perform the copy, but at the same time will take increasingly more time to copy as more machines are added. This is because each machine takes 18 interactions to copy after the initial rack has been created. At the same time, version 2.x requires just 15 interactions, regardless of the number of machines in the rack. It must also be noted that for our testing, we only use four components per machine. With more complex servers, the performance gaps grow even faster as machine are added.



(a)



(b)

Figure 11: Interactions required to complete a KLM task in OpenDC without and with prefabs.

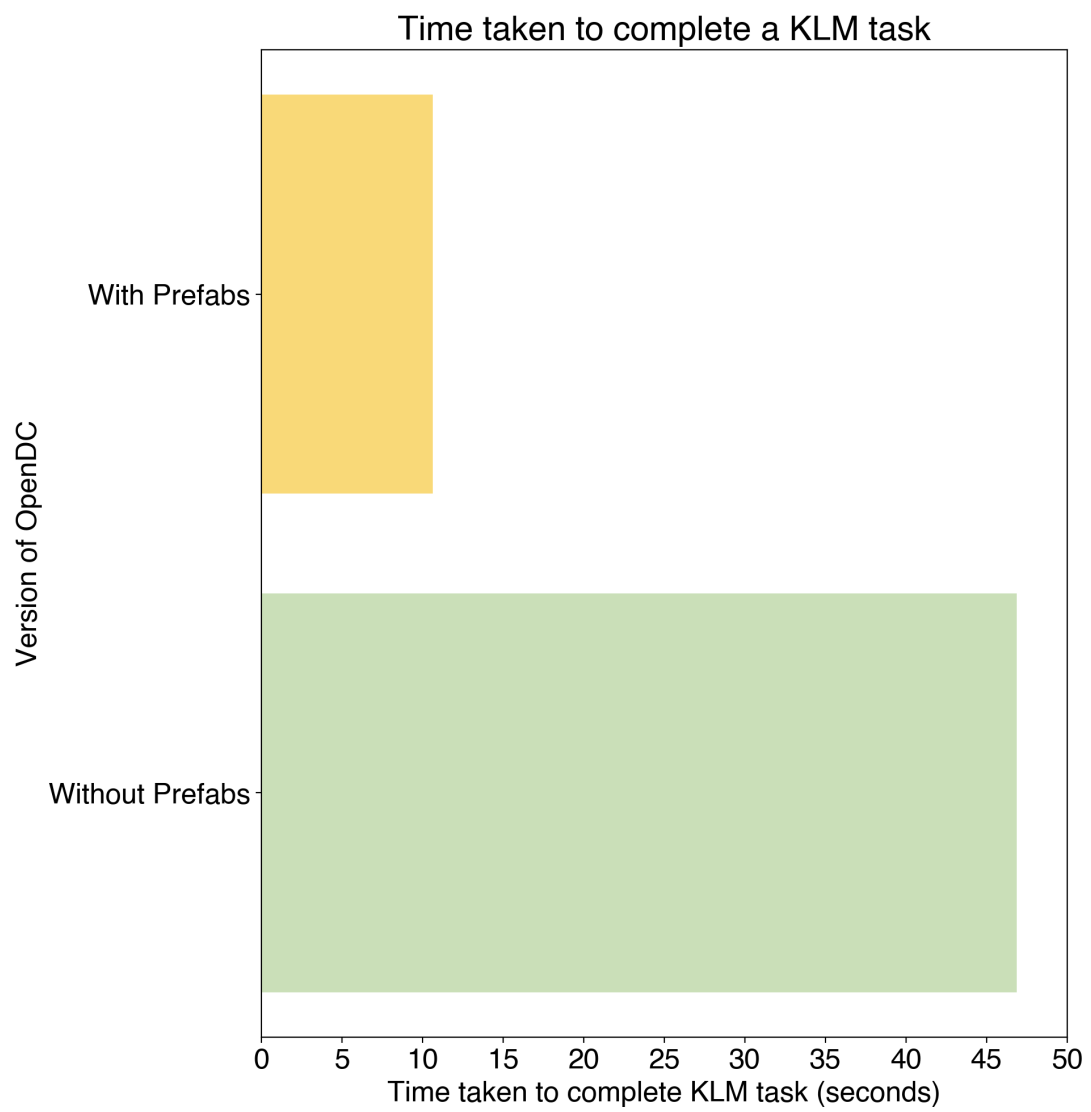


Figure 12: Comparison of time taken to complete a KLM task without and with prefabs.

As a result of the reduced number of interactions required, we can also conclude that the time taken to copy a rack is reduced by the addition of prefabs. Using the values in Table 1, we can calculate the time required for version 1.x to complete a copy in our testing scenario using the below equation:

$$M + 35P + 35B + 35R = 1.35 + 38.5 + 3.5 + 3.5 = 46.85$$

As a result, we can determine that it takes 46.85 seconds for an average user to complete the copy task defined in our testing setup without the use of prefabs.

We then calculate the time taken for version 2.x to complete the task in the same way:

$$M + 7P + 8B + 8R = 1.35 + 7.7 + 0.8 + 0.8 = 10.65$$

From these results, we can see that prefabs would allow an average user to complete the copy task in 10.65 seconds. For this scenario, the use of prefabs leads to $4.4\times$ faster completion time than the conventional approach.

Using this modelling approach, we can represent more diverse and more complex scenarios. For all cases, the repetitive addition of components, which characterizes current datacenter designs (and especially large-scale, homogenous designs) leads to significant performance gains when using prefabs over the conventional approach (without prefabs). This is caused by the repetition of $P - B - R$ cycles, the use of which is linear with the addition of prefabs. Without prefabs, the occurrence of this cycle (which represents moving the mouse and clicking), grows rapidly with each machine that needs to be copied.

Depending on the length of the copying session, it is also possible that the absence of prefabs may cause an increase in the presence of the H action (whereby a user homes their hands on the keyboard or mouse), as the copying session is broken up into smaller sessions. Because copying is much faster with prefabs, this same phenomena may be avoided, as copying sessions would typically be much shorter than by conventional methods.

The factor M (whereby a user mentally prepares for a task) could have a larger effect for prefabs than for conventional approaches, as a user must think about which prefab to copy. However, the reverse is just as likely to be true, with repeated additions of single items potentially requiring much more mental preparation in the conventional approach (e.g. as designers maintain a running count in their head of how many times they have copied an item). Real-world experiments where real human actions are measured for this cognitive task are necessary to effectively test for this effect.

A code-based approach to programming prefabs via a console in the frontend may also lead to performance gains, as commands can be re-used quickly. However, the factor M would scale linearly as the number of repetitions increases, as each line of code would require some mental preparation to execute (e.g. to check for syntax errors, as well as the correctness of the command in terms of desired outcome).

As a result of these experiments and our subsequent analysis, we conclude prefabs show significant promise for improving performance over the conventional, non-prefab-based approach. Furthermore, we expect prefabs to lead to fewer mistakes and overall lower risk of introducing incorrect elements in repeated design-features. We anticipate that a similar speedup and reduction of incorrect elements will become possible in other aspects of OpenDC, as more prefab operations are added to the OpenDC codebase.

6 Conclusion

In this final chapter, we assess the suitability of our solution as a response to the problem presented in Chapter 1. We also discuss the future of this work, exploring ways in which this work could be further developed and improved, and how these ideas align with the vision for OpenDC [18].

6.1 Our Answers to the Research Questions

Datacenter usage is growing fast, and will only continue to grow as more people use products that depend on cloud services. The growth of markets for IoT and game and content streaming will lead to even greater demand for datacenter capacity in the near future. As a result, it remains important that there are people equipped to design these new datacenters, and their expansions. With the extensions made to OpenDC in this thesis, we intend to help broaden the field of people who are equipped to create these designs, allowing this explosive growth to continue.

In this thesis, we have presented a design for a prefab data structure that is capable of expressing a wide variety of common datacenter hardware. Through creating a library of HPC prefabs, we have shown that this data structure is capable of representing common datacenter hardware with a high degree of fidelity. Because of its use of JSON, the data structure is also easily extensible, supporting future work both in terms of the capabilities of OpenDC, and in terms of server design. The JSON basis also provides a hierarchical organisational structure, in which components have a clear and logical place, allowing the structure itself to be easily readable and understandable. As a result, we believe we have sufficiently answered **RQ1**: *“How to design a prefab abstraction that describes important technical and composable features of datacenter hardware to a high degree of fidelity?”*.

In addition to creating a design for prefabs, we have also successfully created a functional prototype for experimenting with the new technologies introduced with the prefabs extension. Our prototype, composed of a Python frontend, a Python command-line interface, and a MongoDB backend, supports many of the operations proposed in our design, and serves as a useful playground for testing additional features that could be added to prefabs. Creating prefabs in an environment based on the structure of OpenDC, without fully implementing the design in OpenDC allows for quick experimentation. This provides the ability to get rapid feedback while avoiding spending time implementing an idea into the broader OpenDC codebase, making it easier to justify attempting a wider range of approaches to the problem. By creating this prototype, and detailing decisions made in the process, we have produced an answer to the question posed in **RQ2**: *“How to implement a prototype of the prefab abstraction?”*.

Lastly, to assess the use of prefabs in practice, we used KLM testing to show that operations on prefabs can provide a significant increase in prototyping speed. We find that KLM testing provides a simple to understand and easy to implement method of testing improvements made to ease of use, which we think will be of value in further extensions of OpenDC. The improvements in performance found in our testing increase as the complexity of the datacenter increases, leading to even larger performance improvements for users tasked with designing massive-scale topologies. Now that the prefab data structure has been implemented in OpenDC, along with an implementation of a desirable operation on

prefabs, other operations can be easily added to the project to further increase usability and prototyping speed. We expect that, with user interface refinements, it may even be possible to further decrease the time taken to design a topology. In addition to validating the use of prefabs by using them to represent existing server hardware from a specific domain, we have proposed a method for doing so that is applicable to other domains. Using information provided by server OEMs to define our domain-specific prefabs is suitable for at least the larger domains within datacenter usage, if not also for smaller and emerging domains such as game streaming. However, we are limited in this approach to the domains that a given OEM chooses to target: smaller and emerging domains may not be considered large enough economically for an OEM to design for, leading to a reduced set of existing servers to model. However, we do find that our approach has been successful for the domain of HPC, producing high-quality representations of existing server models currently offered on the market. With the design of both of these methods of validating our prefab implementation in practice, we have answered **RQ3**: “*How to assess the use of prefabs in practice?*”.

6.2 Future Work

Looking forward, there are several areas of interest that it would be beneficial to visit in future research.

Firstly, there are many more characteristics of datacenter hardware that we could extend OpenDC to represent. We currently do not focus on high-density servers with multiple mainboards (such as blade servers) in each chassis in this thesis. We could also extend rack prefabs to include networking hardware required for the topology. The addition of networking hardware would also be beneficial for modelling *Storage Area Networks* (SANs) and topologies that include nodes comprised of multiple units, such as HP’s Superdome Flex concept¹⁷. Hardware for managing the power distribution throughout a topology would also be an important future expansion. Uninterruptible Power Supplies (UPSes) and Power Distribution Units (PDUs) are critical to the health of datacenter hardware, and take up rackspace. As a result, it would be useful to represent this hardware in OpenDC in the future, as per-rack power limits would be dictated by the presence of such hardware.

Next, the OpenDC user interface could be further refined, to increase functionality and improve usability. A possible addition to the UI could be a powerful text-based console within the browser, so that the creation and manipulation of topologies and prefabs could become even more intuitive, allowing for even faster (and reproducible) prototyping. While the features we have introduced in this thesis are aimed towards a more general, less technically-oriented audience, the above feature would be aimed at users with more experience in datacenter design.

Additionally, we could extend domain-specific prefabs to include more domains. So far, we have covered the domain of High Performance Computing, but there are many more that remain that are extremely relevant to the direction of the industry. Areas such as cloud gaming, or business-critical infrastructure have different requirements of their hardware choices, and so their topologies and thus their prefabs would likely look quite different from what we have explored so far in this thesis. Future work could investigate

¹⁷<http://hpe.com/superdome>

what kinds of hardware is used in these fields, and what useful prefabs would look like in these domains.

Furthermore, prefabs could be extended with more functionality. Currently, prefabs only support creation and deletion, with no defined way of editing prefabs, and no versioning system to track edits across prefab versions. By enabling this possibility, prefabs can become more intuitive to use, with versioning providing even more possibilities for prefab usage. It would become clear if a prefab you are dependent on has been updated, e.g. to support more detail, or a new family of components.

Extending prefabs to support prefabs of varying sizes would also provide benefits. By enabling users to save whole rooms as prefabs, designing and experimenting using OpenDC would get even faster, allowing dozens of racks of machines to be copied in just a few clicks. On the opposite end of the scale, being able to provide small prefabs for chassis would also provide a more intuitive approach for sharing models of individual server nodes.

Lastly, it would be interesting to design a means for OpenDC to simulate user-specified workloads on systems designed by OpenDC itself. OpenDC would be able to leverage a large database of performance data to make component choices based on the workload specified by the user. OpenDC could even be taught how to perform such a task, using Machine Learning with existing datacenter designs as training models. Research could be carried out to determine which hardware characteristics translate best into high performance for certain workloads, leading to objectively performant prefabs designed for those workloads. These research-based prefabs would then serve to validate the prefabs that OpenDC learns to build, and further help develop this functionality. In this way, the system would be designed for the best objective performance at the specified workload, potentially adherent to specified constraints such as financial or power budget. If desired, the level of user interaction could be minimal to none, perhaps only requiring small changes to the design.

References

- [1] P. Andrade, T. Bell, J. Van Eldik, G. McCance, B. Panzer-Steindel, M. C. Dos Santos, S. Traylen, and U. Schwickerath. Review of CERN data centre infrastructure. *Journal of Physics: Conference Series*, 396(PART 4), 2012.
- [2] G. Andreadis, L. Versluis, F. Mastenbroek, and A. Iosup. A reference architecture for datacenter scheduling: Design, validation, and experiments. *Proceedings - International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018*, pages 478–492, 2018.
- [3] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(05):54–63, May 2016.
- [4] T. Bang, N. May, I. Petrov, and C. Binnig. The tale of 1000 Cores: An evaluation of concurrency control on real(ly) large multi-socket hardware. *Proceedings of the 16th International Workshop on Data Management on New Hardware, DaMoN 2020*, 2020.
- [5] A. Bonkoski, R. Bielawski, and J. A. Halderman. Illuminating the security issues surrounding lights-out server management. In *7th {USENIX} Workshop on Offensive Technologies ({WOOT} 13)*, 2013.
- [6] R. Buyya and M. Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency Computation Practice and Experience*, 14(13-15):1175–1220, 2002.
- [7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.
- [8] H. Casanova, A. Legrand, and M. Quinson. SimGrid: A generic framework for large-scale distributed experiments. *Proceedings - UKSim 10th International Conference on Computer Modelling and Simulation, EUROSIM/UKSim2008*, pages 126–131, 2008.
- [9] R. S. Couto, M. E. M. Campista, and L. H. M. Costa. A reliability analysis of datacenter topologies. *GLOBECOM - IEEE Global Telecommunications Conference*, pages 1890–1895, 2012.
- [10] C. Davatz, C. Inzinger, J. Scheuner, and P. Leitner. An Approach and Case Study of Cloud Instance Type Selection for Multi-Tier Web Applications. *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, pages 534–543, 2017.
- [11] T. Dean, J. Andrews, and J. West. *Network+ Guide to Networks*. Cengage, Boston, MA, 8 edition, 2018.
- [12] DigitalOcean. DigitalOcean Marketplace, 2020.

- [13] A. Drapeau, K. Shirriff, J. Hartman, E. Miller, S. Seshan, R. Katz, K. Lutz, D. Patterson, E. Lee, P. Chen, and G. Gibson. RAID-II: a high-bandwidth network file server. In *Proceedings of 21 International Symposium on Computer Architecture*, pages 234–244. IEEE Comput. Soc. Press, 1994.
- [14] M. E. Elsaid and C. Meinel. Multiple virtual machines live migration performance modelling - VMware vMotion based study. *Proceedings - 2016 IEEE International Conference on Cloud Engineering, IC2E 2016: Co-located with the 1st IEEE International Conference on Internet-of-Things Design and Implementation, IoTDI 2016*, 04:212–213, 2016.
- [15] K. Gareis, T. Hüsing, S. Birov, I. Bludova, C. Schulz, and W. B. Korte. e-Skills for Jobs in Europe: Measuring Progress and Moving Ahead. Technical report, European Commission, Bonn, 2014.
- [16] IBM. IBM Power ISA Version 3.0B. Technical report, IBM, 2017.
- [17] IBM Global Technology Services. Prefabricated modular data center—add data center capacity where and when you need it. (September), 2014.
- [18] A. Iosup, G. Andreadis, V. V. Beek, M. Bijman, E. V. Eyk, M. Neacsu, L. Overweel, S. Talluri, L. Versluis, and M. Visser. The opendc vision: Towards collaborative datacenter simulation and exploration for everybody. *Proceedings - 2017 IEEE 16th International Symposium on Parallel and Distributed Computing, ISPDC 2017*, pages 85–94, 2017.
- [19] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. Epema. The Grid Workloads Archive. *Future Generation Computer Systems*, 24(7):672–686, 2008.
- [20] A. Iosup, N. Yigitbasi, and D. H. J. Epema. On the performance variability of production cloud services. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2011, Newport Beach, CA, USA, May 23-26, 2011*, pages 104–113. IEEE Computer Society, 2011.
- [21] B. Javadi, D. Kondo, A. Iosup, and D. H. J. Epema. The failure trace archive: Enabling the comparison of failure measurements and models of distributed systems. *J. Parallel Distributed Comput.*, 73(8):1208–1223, 2013.
- [22] A. T. Kabakus and R. Kara. A performance evaluation of in-memory databases. *Journal of King Saud University - Computer and Information Sciences*, 29(4):520–525, 2017.
- [23] K. E. Kushida, J. Murray, and J. Zysman. Cloud Computing: From Scarcity to Abundance. *Journal of Industry, Competition and Trade*, 15(1):5–19, 2015.
- [24] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf. NIST Cloud Computing Reference Architecture. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, 2011.

- [25] G. Macatee, M. Shirer, P. Maguranis, and S. Lagana. Worldwide Server Market Revenue Declined 6.0First Quarter of 2020, According to IDC. Technical report, International Data Corporation, 2020.
- [26] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, February 2020.
- [27] S. A. Mokhtar, S. H. S. Ali, A. Al-Sharafi, and A. Aborujilah. Cloud computing in academic institutions. *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication, ICUIMC 2013*, 2013.
- [28] A. Newell, S. K. Card, and T. P. Moran. The Keystroke-Level Model for User Performance Time with Interactive Systems. *Communications of the ACM*, 23(7):396–410, 1980.
- [29] B. Pring, R. H. Brown, A. Frank, S. Hayward, and L. Leong. Forecast : Sizing the Cloud ; Understanding the Opportunities in Cloud Services. *Gartner*, (March 2009):13, 2009.
- [30] SURF. Description of the Lisa system, 2020.
- [31] W. Torell. TCO Analysis of a Traditional Data Center vs. a Scalable, Prefabricated Data Center Revision 1. pages 1–7.
- [32] W. Torell. Types of Prefabricated Modular Data Centers. *APC White Paper*, 165(2):13, 2017.
- [33] A. Uta, A. Custura, D. Duplyakin, I. Jimenez, J. S. Rellermeier, C. Maltzahn, R. Ricci, and A. Iosup. Is big data performance reproducible in modern cloud networks? In R. Bhagwan and G. Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 513–527. USENIX Association, 2020.
- [34] V. van Beek, G. Oikonomou, and A. Iosup. Portfolio scheduling for managing operational and disaster-recovery risks in virtualized datacenters hosting business-critical workloads. In A. Iosup, R. Prodan, A. Uta, and F. Pop, editors, *18th International Symposium on Parallel and Distributed Computing, ISPDC 2019, Amsterdam, The Netherlands, June 3-7, 2019*, pages 94–102. IEEE, 2019.
- [35] L. Versluis, R. Mathá, S. Talluri, T. Hegeman, R. Prodan, E. Deelman, and A. Iosup. The workflow trace archive: Open-access data from public and private computing infrastructures. *IEEE Trans. Parallel Distrib. Syst.*, 31(9):2170–2184, 2020.
- [36] L. Versluis, M. Neacsu, and A. Iosup. A trace-based performance study of autoscaling workloads of workflows in datacenters. In E. El-Araby, D. K. Panda, S. Gesing, A. W. Apon, V. V. Kindratenko, M. Cafaro, and A. Cuzzocrea, editors, *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018, Washington, DC, USA, May 1-4, 2018*, pages 223–232. IEEE Computer Society, 2018.

- [37] M. A. Voinea, A. Uta, and A. Iosup. POSUM: A portfolio scheduler for mapreduce workloads. In N. Abe, H. Liu, C. Pu, X. Hu, N. K. Ahmed, M. Qiao, Y. Song, D. Kossmann, B. Liu, K. Lee, J. Tang, J. He, and J. S. Saltz, editors, *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 351–357. IEEE, 2018.
- [38] E. Wittern, P. Suter, and S. Rajagopalan. A look at the dynamics of the JavaScript package ecosystem. *Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016*, pages 351–361, 2016.
- [39] N. Yigitbasi, M. Gallet, D. Kondo, A. Iosup, and D. H. J. Epema. Analysis and modeling of time-correlated failures in large-scale distributed systems. In *Proceedings of the 2010 11th IEEE/ACM International Conference on Grid Computing, Brussels, Belgium, October 25-29, 2010*, pages 65–72. IEEE Computer Society, 2010.
- [40] G. Zheng, G. Kakulapati, and L. V. Kalé. BigSim: A parallel simulator for performance prediction of extremely large parallel machines. *Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2004 (Abstracts and CD-ROM)*, 18(C):1105–1114, 2004.

A Appendix: Data from KLM Tests

version 1.x		version 2.x	
Interaction	Time taken (seconds)	Interaction	Time taken (seconds)
Move to target tile	1.1	Move mouse to target rack	1.1
Click	0.1	Click	0.1
Move to "start rack construction" button	1.1	Click	0.1
Click	0.1	Move mouse to "save this rack to prefab" button	1.1
Move back to target tile	1.1	Click "save this rack to prefab"	0.1
Click	0.1	Move mouse to "back to room"	1.1
Move to "stop rack construction" button	1.1	Click "back to room"	0.1
Click	0.1	Move mouse to "load a rack from a prefab"	1.1
move to newly made rack	1.1	Click "load a rack from a prefab"	0.1
Click	0.1	Move mouse to target prefab	1.1
Move mouse to "add machine" button	1.1	Click	0.1
Click	0.1	Move mouse to "confirm import" button	1.1
Move mouse to "add machine" button	1.1	Click	0.1
Click	0.1	Move mouse to desired rack location	1.1
Move mouse to "add machine" button	1.1	Click	0.1
Click	0.1		
Move mouse to machine one	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "GPU" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "memory" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "storage" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "back to rack" button	1.1		
Click	0.1		
Move mouse to machine two	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "GPU" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "Memory" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "Storage" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "back to rack" button	1.1		
Click	0.1		
Move mouse to machine three	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "GPU" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "memory" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "storage" tab	1.1		
Click	0.1		
move mouse to "add" button	1.1		
Click	0.1		
Move mouse to "back to rack" button	1.1		
Click	0.1		
M + 35P + 35B + 35R		M + 7P + 8B + 8R	
= 1.35 + 38.5 + 3.5 + 3.5		= 1.35 + 7.7 + 0.8 + 0.8	
= 46.85 seconds		= 10.65 seconds	

Table 4: Results of KLM testing before and after the addition of prefabs.