Disaggregated Data Centres – Simulation & Analysis of IT & Network Resource Allocation Algorithms

Ashish Tibrewal

April 2016

Project Thesis submitted in support of the Degree of Master of Engineering in Computer Science and Electronics

Department of Electrical & Electronic Engineering
University of Bristol

Abstract

Data centres are under severe pressure to meet the current globally growing demands of cloud, big data, mobile, and social collaboration applications. Yet, present data centres are built with conventional architectures in mind that can take days or weeks to provision new services and typically run with low server utilisation and efficiency, and limited flexibility incurring higher capital investments and driving up operational costs. To tackle this, modern data centres require a different approach – the disaggregated architecture. Disaggregated architectures are systems in which there are segregated pools of resources that are decoupled and can be allocated on demand, therefore, increasing the utilisation and efficiency. At the core of a disaggregated architecture lies a management framework that controls all the resource pools and creates a wide range of logical, virtual systems based on workload-specific demands. The modular arrangement in a disaggregated architecture also allows for statistical time division multiplexing (STDM), a technique used for dynamic bandwidth allocation (DBA) making it possible for disaggregated data centres to cope with high network demands. The application of dynamic bandwidth allocation in disaggregated architectures makes it possible to cover peak network demands, whilst not under-utilising the same resources during non-peak conditions, thus ensuring optimal utilisation at all times. Disaggregation can significantly improve the scope for scalability, flexibility and versatility. Efficient resource allocation and workload scheduling is a fundamental requirement in data centres, particularly those with a disaggregated architecture, and any high-performance computing (HPC) system, be it, a cluster, grid or cloud computing platform. Globally optimal algorithms can yield better performance, increase the overall compute density and the number of requests served, i.e. virtual machines/systems created.

Declaration & Disclaimer

I declare that the work in this report was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this report which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the report, other than referenced material, are those of the author.

Author

Date

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my project supervisor Dr. Georgios Zervas of the Faculty of Engineering at the University of Bristol for his continuous support, guidance and motivation. The door to Prof. Zervas' office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this thesis to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would like to thank the experts who were involved in the poster assessment for this research project: Dr. Geoff Hilton and Dr. Nick Simpson. Without their passionate participation, input and feedback, the poster presentation would not have been a success.

Finally, I must also express my very profound gratitude to my parents and to my elder brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

API Application programming interface

ASIC Application-specific integrated circuit

BFS Breadth-first search

L1 L2 L3 Level 1, Level 2 & Level 3

CDF Cumulative distribution function

CPU Central processing unit

CR Contention ratio

CRAC Computer room air conditioning

DBA Dynamic bandwidth allocation

DCN Data center network

DIMM Dual in-line memory module

DRAM Dynamic random-access memory

FIFO First-in first-out

FPGA Field-programmable gate array

GPU Graphics processing unit

GWh Gigawatt hour

HDD Hard disk drive

HPC High performance computing

IC Integrated circuit

ICT Information & communications technology

I/O Input/output

IT Information technology

LED Light emitting diode

mBFS Modified breadth-first search

NIC Network interface controller

OPCIe Optical peripheral component interconnect express

PCIe Peripheral component interconnect express

PIC Photonic integrated circuit

PUE Power usage effectiveness

RSA Rack Scale Architecture

RU Rack unit

SRAM Static random-access memory

SSD Solid-state drive

SSL Secure sockets layer

STDM Statistical time division multiplexing

TCO Total cost of ownership

ToB Top-of-blade

ToR Top-of-rack

YAML YAML Ain't Markup Language

1 Introduction

A data centre is a facility that houses information and communications technology (ICT) infrastructure that is used to process, communicate, and store digital data. Traditional data centres follow the *aggregated server-centric architecture*, one in which blades (or drawers) contained in racks are setup as servers, each with a certain amount of compute resources (including hardware accelerators) such as CPUs and GPUs, memory resources such as SRAMs (in the L1, L2 and L3 cache) and DIMMs containing DRAMs, and large capacity secondary storage such as HDDs and SSDs. Server-centric architectures have a relatively static computing infrastructure with a predefined amount of servers.

This thesis investigates a different design approach to data centres – the *disaggregated resource-centric architecture*, one in which there are modular pools of compute, memory, storage and network resources interconnected using a communication network. The modular arrangement in a disaggregated architecture allows, amongst other advantages, statistical multiplexing (a technique used for dynamic bandwidth allocation), which markedly improves performance. Other benefits of disaggregation include improved operational efficiency through increased resource utilisation and interoperability, accelerated service delivery, shared cooling, improved power management and lower total cost of ownership (TCO). Disaggregation makes it possible for data centres to cope with high demands and varying workloads whilst ensuring that the same resources are not under utilised during non-peak conditions. Disaggregation also enables resource-specific upgrades that can markedly improve the scope for scalability, a factor that needs to carefully considered when designing and deploying new data centres. Using a software management framework in a disaggregated architecture allows a wide range of virtual systems to be built on demand. These management frameworks not only enable (rack-wide) resource and policy management, but also provide standard firmware and software APIs that expose the hardware resources to the orchestration layer via a standard interface. [1] Disaggregation decouples services from the underlying infrastructure; this decoupling not only offers more flexibility when designing new data centres, but also makes it possible to rapidly re-architect existing data centres. This is particularly useful when upgrading or scaling only a part of the infrastructure; instead of having to re-architect the entire system due to this change, only the components that need to be upgraded are modified. In this ever-changing technology generation, this flexibility is vital to data centres.

Disaggregated data centres can be configured and laid out in several different ways, each configuration and the layout of resources can have a distinctly different performance; these configurations are discussed in detail further in this thesis. Amongst other challenges, the major challenge with disaggregated architectures is the requirement of super-fast interconnects. Traditional interconnect technologies such as Ethernet, InfiniBand, PCI Express (PCIe), and even super-fast optical fibre technology is not enough to cope with the latency and bandwidth requirements similar to those available on on-board connections. To overcome these issues, disaggregated architectures require advanced silicon photonics[1]

[1] *Silicon photonics* is an emerging technology that utilises silicon to make optical devices that can be integrated into and/or stacked up alongside standard integrated circuits (ICs). [5] [4]

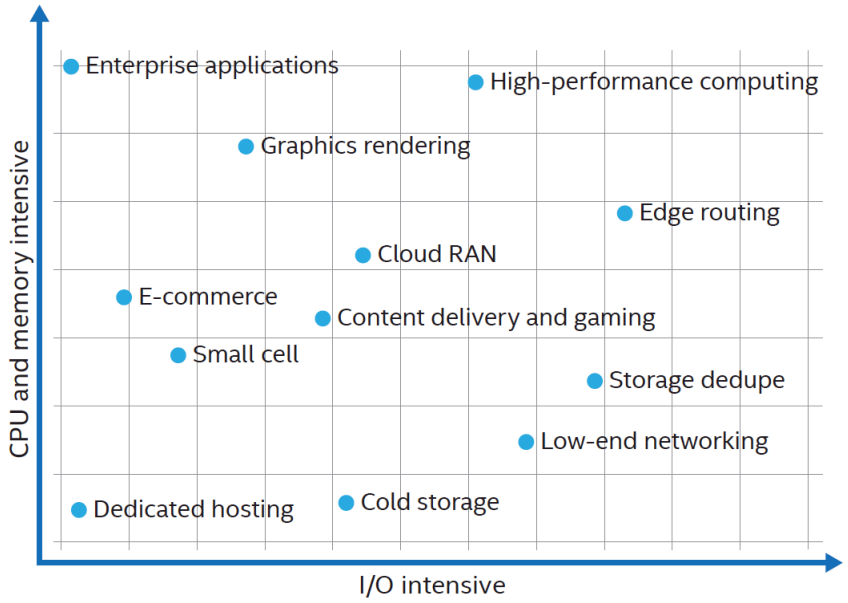


Figure 1: *Diversity of data centre workloads illustrating the disparity of server provisioning required to optimise servers for specific workloads. (Image: Tencent & Intel Corporation)* [2]

technology – a technology which integrates a hybrid silicon laser and electronic components onto the same die. Since, electrical interconnections in computer motherboards are close to being maxed out physically and electronically, the need to replace electrons with photons for inter-device communication is essential for disaggregation to be successful. The appropriate application of silicon photonics and advanced optical interconnect technology can remove connectivity bottlenecks in data centres – this is particularly crucial for data centres providing real-time services. Current technologies use discrete electronics and optics to switch between optical and electrical signals similar to that described by the Fibre-Optic Association. Electrical signals enter the transmitter and are converted into optical signals (using LEDs and/or lasers). These optical signals travel through the connected fibre-optic channels and once it reaches its destination, the exact inverse process occurs at the receiver – the light detector at the receiver converts the optical signals into electrical impulses. Although this technology has been around for many years, and it works well, it is expensive and labour intensive, primarily due to all the individual electronic components such as amplifiers, drive optics, lasers, and microchips, etc. Due to this, researchers have investigated silicon as an alternative to using all these separate components, but studies have found that silicon has several undesirable characteristics. "Silicon is a poor candidate for photonic applications because its electronic structure has an 'indirect band gap', making it a poor light emitter. This means that when an electron and hole combine in silicon, the resulting energy released is more likely to be emitted as vibrational energy, or phonons rather than as photons," as explained by Mario Paniccia, director of Intel's photonics technology laboratory. [3] Another issue that was discovered

during these studies was that silicon lacks an electro-optic effect. In simple terms, this means that when modulating optical signals, silicon is not known for its ability to change optical properties in response to an electrical field. [3] In late 2013, Fujitsu, a Japanese multinational information technology equipment and services company, in collaboration with Intel Corporation, announced that they would be releasing their first product using silicon photonics – the Optical PCI Express (OPCIe) server in early 2017. [4] Although silicon photonics technology is in its early development phase, its potential cannot be underestimated.

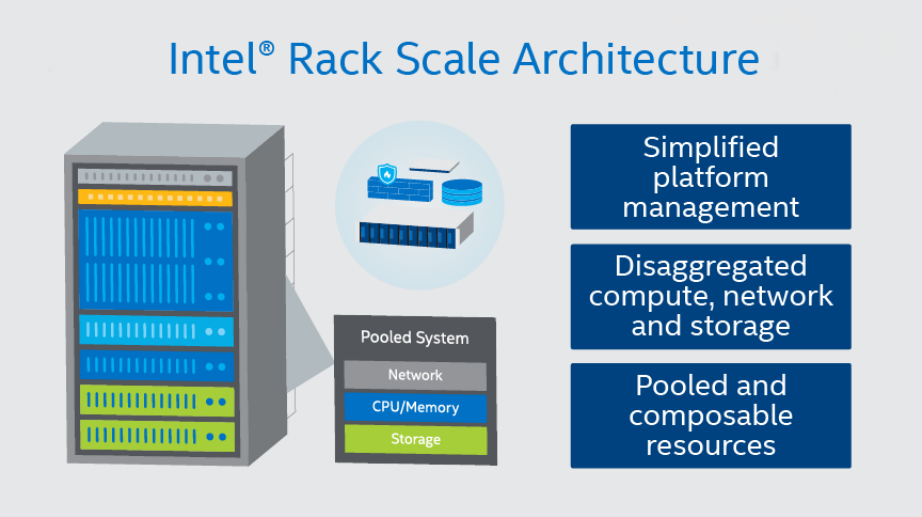


Figure 2: *Intel's proprietary disaggregated architecture technology – Intel ® Rack Scale Architecture (RSA). (Image: Intel Corporation)* [1]

Industry leaders such as Intel Corporation, IBM and Ericsson are amongst the first few to have already taken the initiative to generate the next-generation disaggregated data centre technology. Intel's proprietary disaggregated architecture solution, the Rack Scale Architecture (RSA), shown in Figure 2, was unveiled in late 2013; it promises to be highly efficient and scalable. Early adoption of Intel's Rack Scale Architecture by Ericsson, to build it's next-generation hyperscale cloud platform, known as the Ericsson Cloud System in the form of Ericsson HDS 8000 server addresses today's growing challenges for reliable and robust connectivity, real-time data, network efficiency and on-demand service delivery with integrity, governance, and automation. [6]

The primary research area covered in this thesis is based on resource allocation algorithms in disaggregated architectures. Several different resource allocation algorithms were developed and simulations were carried out for different types of disaggregated architectures. The performance of each algorithm was analysed based on different performance metrics such as blocking probability, resource utilisation, latency, etc. In summary, these algorithms can be reduced to the standard multidimensional bin-packing problem. Resource allocation algorithms play a crucial role in determining the overall performance of a data centre; globally optimal algorithms that are efficient can significantly increase

resource utilisation, improve the overall compute density and the performance of data centres in general. Detailed discussions of the algorithms developed are provided in the Resource Allocation Algorithms section and an analysis on their performance is provided in the Results section.

2 Data Centre Architectures

2.1 Aggregated Server-Centric Architecture

As discussed previously, traditional server-centric data centres are built using hundreds of servers with the ability for each to function independently. Each of these servers contain a predefined amount of compute, memory, storage and network resources, creating a static infrastructure. Figure 3 shows the aggregated architecture of a conventional server-centric data centre in which each rack consists of multiple server-blades, each with a predefined amount of compute, memory and storage resources. The network architecture of data centres consisting of the core, aggregate and access layers including its switch hierarchy is discussed in detail in the Data Centre Network (DCN) Architecture section.[1]

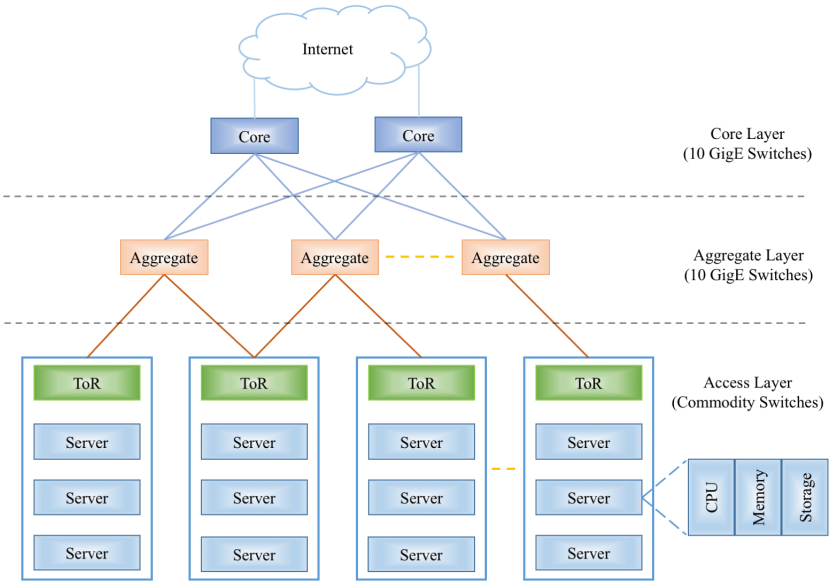


Figure 3: *Aggregated architecture of a conventional server-centric data centre.*

Since most aggregated server-centric architectures are inflexible, existing data centres that have adopted this approach contain servers that are not optimally configured for their purposes – resulting in waste, operational inefficiency and low deployment density. For example, in compute-dense applications, unused memory and hard disk drive slots can negatively affect computing density; in memory-dense applications unused expansion and hard disk drive slots waste server space that could be used for more memory; and in storage-dense applications CPUs and memory might be overprovisioned. [2] Specialised servers are created to mitigate this issue – resulting in new challenges to server resource management, day-to-day maintenance, and overall data centre operations. All these factor combined introduce an additional layer of complexity to data centre management and maintenance. [2] CPU performance has been doubling every two years, whereas memory performance and storage capacity have been advancing at a much slower rate; misaligned technology advances such as these produce big gaps in server optimisation making it

[1] As this thesis aims to investigate the IT and network infrastructure of data centres, and allocation algorithms; other design considerations such as mechanical & electrical infrastructure, power supply management, cooling, security, and environmental issues, etc. have not been discussed in detail.

difficult to upgrade to more efficient processors, memory and storage without unnecessarily discarding still-useful resources. [2]

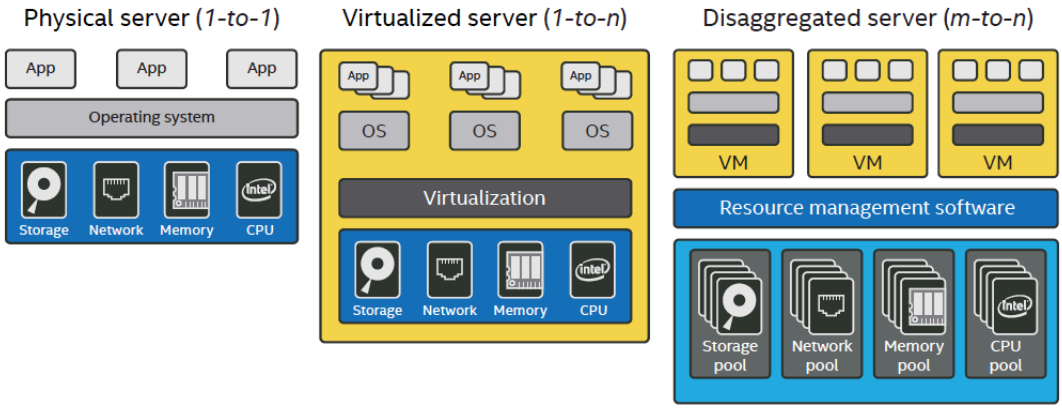


Figure 4: *Physical server (1-to-1), virtualised aggregated server (1-to-n), and software-managed virtualised disaggregated server (m-to-n). (Image: Tencent & Intel Corporation)* [2]

Virtualisation is a good midpoint phase in server optimisation that has a firm footing in data centres today. There is no doubt that virtualisation and cloud implementations have significantly improved resource utilisation in data centres, however, CPUs and memory in such environments are still often underutilised due to the static infrastructure and granularity of server resources. [2] Figure 4 illustrates various models of data centre environments. It can be seen that to increase capacity in the 1-to-1 single-server environment, additional servers are required, thus leading to underutilisation. The 1-to-n virtualised server model uses a single physical server that can divide its physical resources across multiple virtual machines. This certainly does reduce the amount of idle resources but does not completely eliminate inefficiencies. In comparison to these, the m-to-n disaggregated server provides even greater efficiencies by only allocating resources based on the requirements. Cloud services remain virtualised at the software level, but resource-pooled servers provide further virtualisation at the hardware level. [2] The hardware-level virtualisation is provisioned by a resource management software; this is discussed in detail in the following section. The 1-to-n virtualised server and m-to-n disaggregated server approaches are not contradictory, but complementary; resource-pooled servers can continue to run virtualisation and cloud server software on logical servers and create virtual machines for end-users. [2]

Figure 5 shows a plot of the memory-to-CPU and disk-to-CPU consumptions for tasks in Google's data centre; it can be seen that resource requirement is spread over more than three orders of magnitude with approximately 70% of tasks requiring far less memory as compared to CPUs; a similar trend follows for storage requirements. [7] Even with support for virtualisation, such diverse requirements undoubtedly lead to underutilisation of resources. To address the new challenges faced by data centres today and mitigate the shortcomings of the conventional aggregated server-centric architecture, new designs such

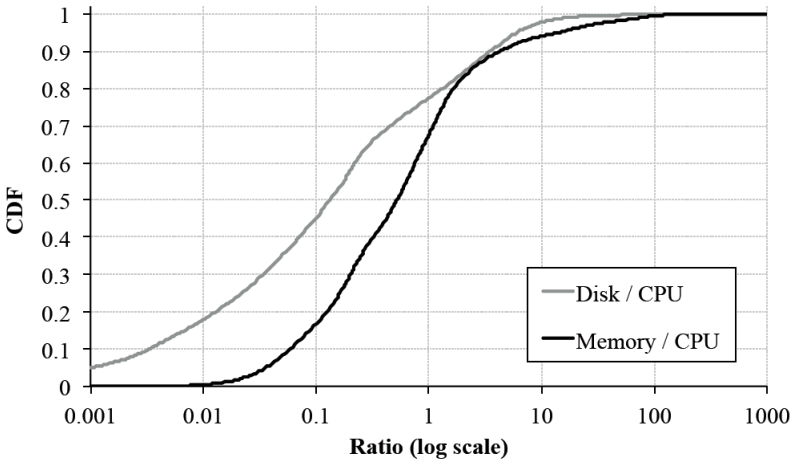


Figure 5: *Cumulative distribution function of relative disk/memory capacity demand to CPU usage for tasks in Google's data centre.* [7]

as the disaggregated resource-centric architecture, that are more efficient and versatile need to be considered and implemented.

2.2 Disaggregated Resource-Centric Architecture

With data volumes and internet traffic growing exponentially every year and business needs changing dramatically, traditional data centres are too rigid to cope with this change and complexity. Ninety percent of the world's data has been created in the last five years and more than 2.5 quintillion bytes are added to this everyday. [2] The worldwide mobile data traffic is also projected to grow at a compounded rate of 57% for the next four years and is expected to reach a throughput of 24.3 exabytes per month by 2019. [2] To cope with this change and trends in data growth, data centres designs need a different approach. Disaggregated data centres are software-defined data centres where the virtualisation principles of abstraction, pooling and automation are not only applied to transform the compute layer but also extended to all other data centre domains, viz. storage, network, security and availability, etc. With such virtualisation, the management is completely automated by software. The benefits of this virtualisation include improved efficiency, control, agility and choice. With all services virtualised, resource utilisation and automation can be dramatically increased, leading to higher capex and opex savings across the entire data centre. In a software-defined data centre, the management layer is responsible for controlling the pools of compute, memory, network and storage; dynamically allocating these resources to apps and services. [1] App provisioning, availability, security and compliance are all managed by policy-based automation resulting in new levels of business agility and operational control. Software-defined data centres enable businesses to build non-vendor-specific infrastructure, therefore, increasing the scope for change, adaptability and scalability. It is important to realise and understand that with the exponentially growing need of IT capacity for businesses, it is going to be al-

most impossible for these businesses to financially sustain this growth without a change in approach. A methodology to achieve steadily improving hyperscale performance is to adopt and continuously iterate through the industrialisation cycle, one that comprises five major steps; these include standardisation, combination (and consolidation), abstraction, automation, and governance. [8]

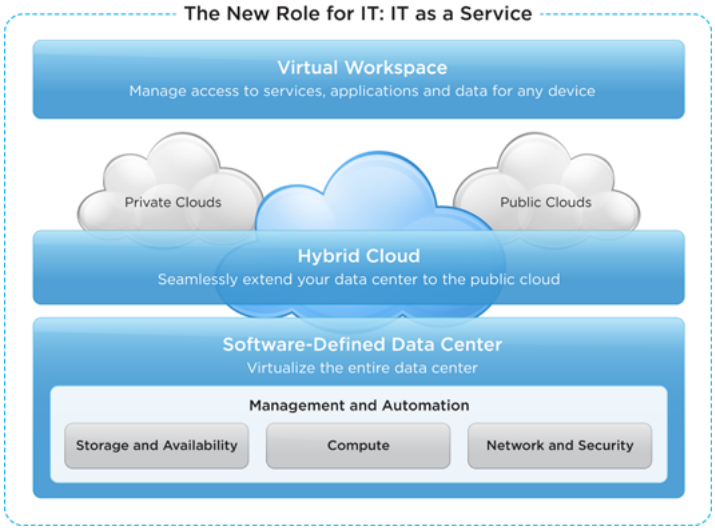
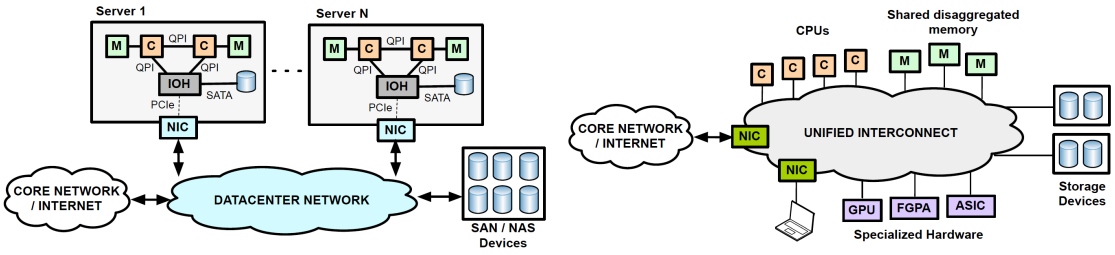


Figure 6: *Software-defined disaggregated data centre IT stack that can be used to build hybrid clouds. Hybrid clouds are a combination of on-premises private clouds and off-premises public clouds. (Image: VMware)* [9]

Disaggregation decouples each hardware unit, i.e. each individual CPU, memory, storage, or network unit, from every other hardware unit; this can be beneficial when managing extreme (and diverse) workload conditions that present day data centres have to handle. Studies have shown that data centres globally consume over 100 GWh per year and is expected to exceed 130 GWh by the end of 2016. [2] As measured by the power usage effectiveness (PUE) index defined by The Green Grid, CRAC (Computer room air conditioning) units alone can consume as much as half of a data centre's power needs. [2] Disaggregated data centres also allow for better and more efficient power management and cooling. These factors need to be considered carefully since they directly affect the performance of data centres.

Figure 7 illustrates the differences between aggregated server-centric and disaggregated resource-centric architectures. It can be seen that in comparison to resource-centric architectures, server-centric architectures don't provide much scope for customisation. Due to the decoupling of resources, disaggregated architectures can be fully customised and if needed, it can support custom ASICs (Application-specific integrated circuit) and FPGAs (Field-programmable gate array). Disaggregation of I/O devices is fairly straightforward, but the major challenge lies in memory disaggregation as it requires low latency, high bandwidth links that can support speeds and capacity comparable to those achieved in blades with on-board memory; ensuring this can guarantee that disaggregation would



(a) Current aggregated architecture. [7] *(b) Future disaggregated architecture.* [7]

Figure 7: *Comparison between aggregated server-centric architecture and disaggregated resource-centric architecture.*

not negatively affect the performance of data centres. Figure 8 shows several rack-level configurations possible in disaggregated architectures. The rack on the left is configured to be completely homogeneous with homogeneous CPU blades, whereas the rack on the right is configured to be heterogeneous with heterogeneous CPU and memory blades, and homogeneous storage blades.

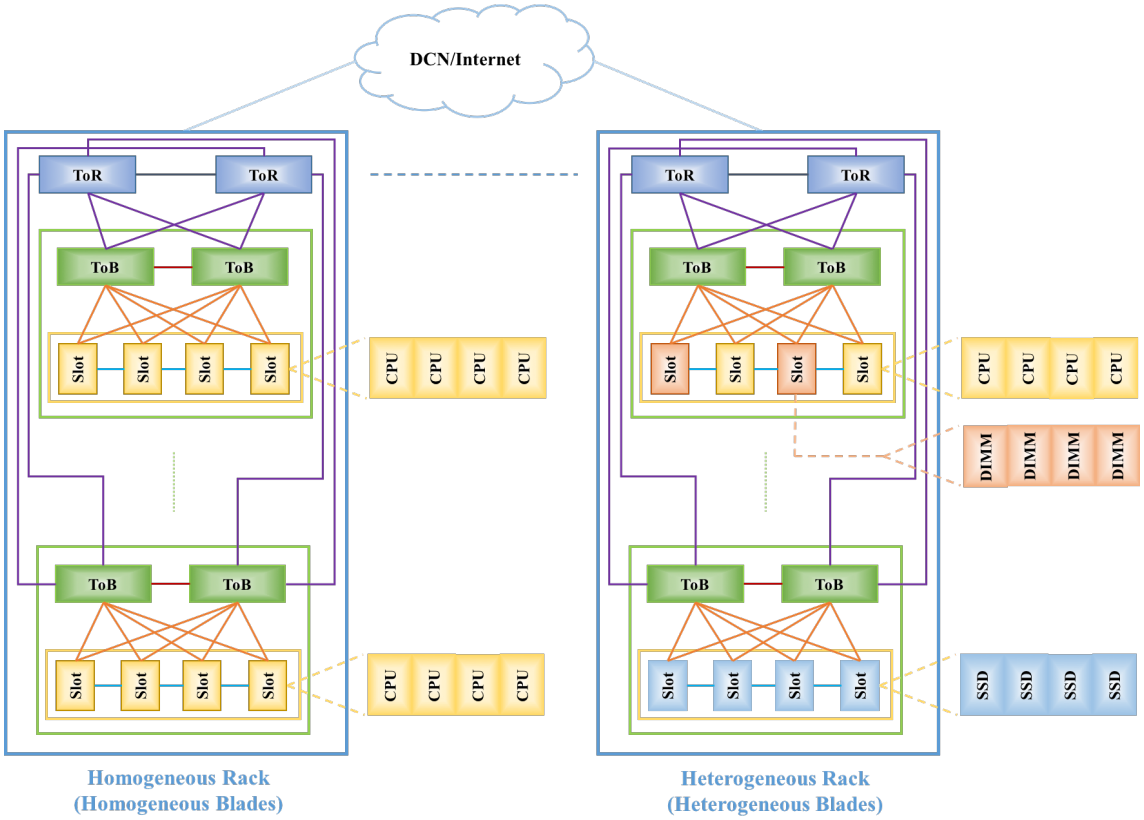


Figure 8: *Disaggregated architecture of a resource-centric data centre illustrating different rack configurations along with the intra-rack switch hierarchy. The large blue boxes represent racks; within each, the green boxes represent blades that contain several configurable slots. The dotted blue line shows an example of an intra data centre connection; in this case, it is a connection between ToRs from different racks.*

As stated in a paper by Ericsson, the four-pillars of disaggregated architectures include

a (software) manager for multi-rack management, pooled system, scalable multi-rack storage, and efficient configurable network fabric. [8] The software management layer exposes hardware, firmware, and software APIs that enable management of resources and policies across the entire data centre. [8] As discussed previously, the pooled system enables the creation of virtual machines using pooled-resources that include compute, memory, storage, and network based on the workload requirements. The scalable multi-rack storage includes Ethernet connected storage that can be loaded with storage algorithms to support a wide range of uses. [8] The efficient configurable network fabric comprises of the network hardware such as NICs (Network interface controller), optical interconnect and management that support a wide range of cost-effective network topologies. Most network designs include the standard top-of-rack (ToR) switch but can be extended to utilise distributed switches that remove several levels of switch hierarchy. [8].

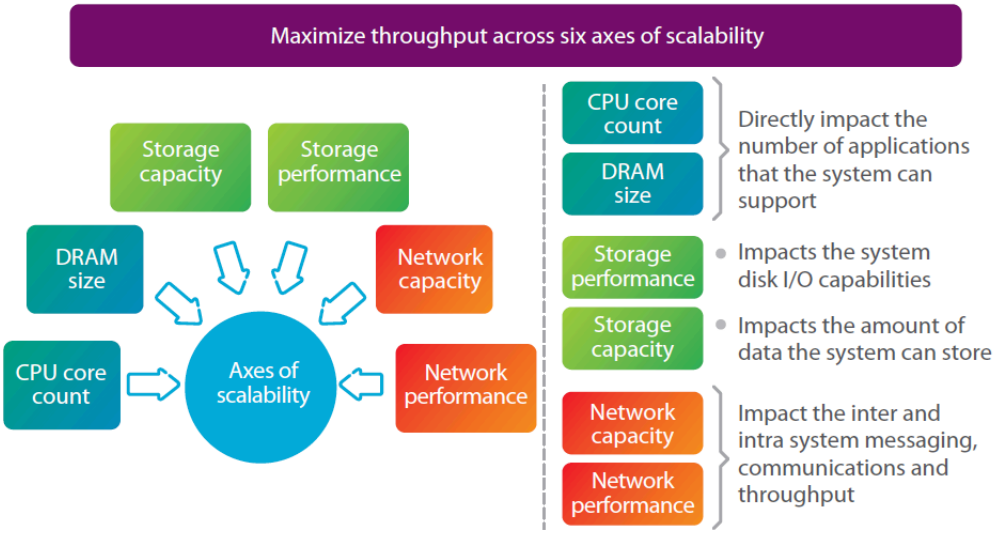


Figure 9: *The six axes of scalability. (Image: Ericsson)* [8]

Resource-pooling requires a new software stack to manage resource discovery, resource allocation, and resource monitoring, etc. that contains several layers, each providing a distinct functionality. [2] The most crucial layer is the management layer; it is responsible for providing APIs to manage low-level pooled resources and provides APIs that expose the middle-layer software offered by public cloud vendors such as VMware, Microsoft, etc. and OpenStack, which is an open-source software platform for cloud computing. [2] It is important to note that these software stacks do not alter operations of any third-party business software, such as Microsoft's Azure, VMware's VSphere[1], or privately developed hypervisors and management stacks – resource-pooled servers are only re-architected in its hardware design and it is completely transparent from a software perspective. [2]

Disaggregation breaks the three-to-four year refresh cycle and can help facilitate incremental resource-specific upgrades that is straightforward and non-disruptive. Figure 9 shows the six axes of scalability; each of these factors need to be considered carefully as

[1] *Microsoft Azure* is Microsoft's proprietary cloud computing platform; *VMware VSphere* is VMware's proprietary cloud computing platform.

it directly affects the performance of data centres. These six factors include CPU core count, DRAM size, storage capacity, storage performance, network capacity, and network performance. [8]. The CPU core count and DRAM size directly impact the number of applications the system can support. The storage capacity and storage performance impacts the amount of data the system can store and it's I/O capabilities respectively. The network capacity, i.e. bandwidth, and network performance, i.e. latency, impact the inter and intra system messaging, communication and throughput of the system. The differential rate of change in these six components drive the need to consider them separately as part of the overall data centre life cycle management. [8]. Along with the disaggregated architectural approach, the implementation of a single system architecture enables organisations to systematically address all major workloads. In summary, a scalable unified infrastructure management architecture can increase utilisation by enabling the allocation of shared resources depending on the workload requirement without needing additional data centre hardware. [8]

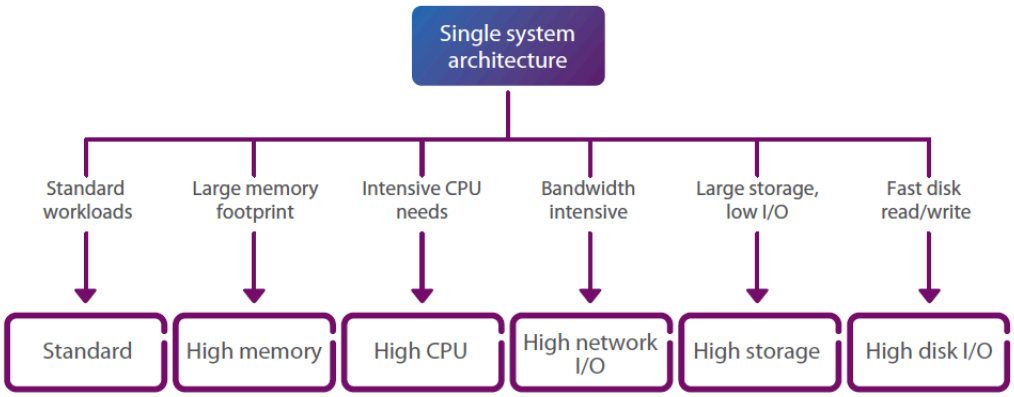


Figure 10: *A single system architecture addresses all major workloads. (Image: Ericsson)* [8]

The combination of disaggregated hardware architectures with advanced optical interconnect and silicon photonics technology is capable of removing the traditional distance and capacity, i.e. latency and bandwidth, limitations of electrical connections. As discussed previously, the major bottleneck in current data centres lies in its network interconnect; this especially poses a big challenge in disaggregated architectures since resource pooling makes it significantly more difficult to satisfy the latency and bandwidth requirements; and only gets more difficult as the resource pools are placed further apart from each other. To mitigate these issues, advanced optical interconnect and silicon photonics technology need to deployed. Conventional commodity electronics technology used in interconnects that span over 10 cm to 2 km need to be replaced by high capacity and low latency optical links. Intel recently announced their protocol-neutral MXC connector and ClearCurve optical fibre that promises to offer high speeds over long distances at a relatively low cost and can be used throughout a data centre over existing technologies as well as Terabit Ethernet. [2] One of the major challenges that the silicon photonics

technology faces today is the creation of efficient silicon light sources that can be used in silicon fabricated PICs (Photonic integrated circuits) and hybrid integrated circuits, i.e. chips containing both electronics and photonics; making the technology affordable and commercially available in large quantities are amongst a few other challenges that need to be addressed. As stated by Intel's former senior vice president, Pat Gelsinger, "Today, optics is a niche technology. Tomorrow, it's the mainstream of every chip that we build." [4]

2.3 Data Centre Network (DCN) Architecture

Data centre networks (DCNs) hold a crucial role in a data centre as it interconnects all of the data centre resources together. In order to handle the growing demands of cloud computing, high bandwidth low latency communication, and memory I/O performance dependent applications, DCNs need to be highly efficient and scalable – most of today's data centres are constrained by the interconnection network. [10] Amongst several other well-know DCN architectures, there are currently three major types – Three-tier DCN, Fat tree DCN, and DCell.

Most DCN designs are based on a proven layered approach, one that has been tested and improved over the past several years in some of the largest data centre implementations in the world. [10] The legacy layered DCN approach forms the basic foundation of data centre design that seeks to improve scalability, performance, flexibility, resiliency and maintenance. [10] Figure 11 shows the basic structure of a layered DCN design. The layered

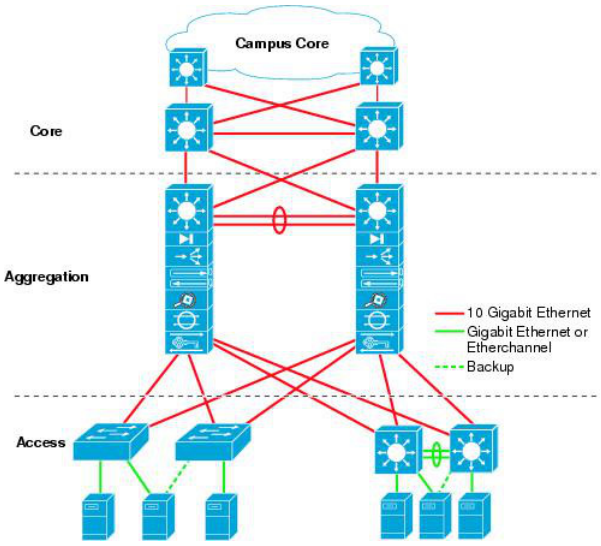


Figure 11: *Basic structure of a layered DCN consisting the core, aggregate and access layer switches. (Image: Cisco Systems)* [10]

three-tier DCN design consists of three major layers – the core, aggregation and access layers. The core layer (layer 3) provides the high-speed packet switching backplane for all flows going in and out of the data centre and connectivity to multiple aggregate modules. [10] The aggregate layer (layer 2) provides important functions such as service module

integration, Layer 2 domain definitions, spanning tree processing, etc. [10] The aggregate modules manage and control server-to-server multi-tier traffic flows and provide services such as firewall, SSL offload, network analysis, and server load balancing to optimise and secure applications. [10] The access layer (layer 1) is where the servers physically connect to the (internal) network. It consists of modular switches, fixed configuration 1 or 2RU[1] switches, and internal blade server switches. [10] Other DCN architectures that include the fat tree and DCell aim to tackle problems faced by the three-tier DCN architecture, such as cross-section bandwidth and over-subscription, etc. by deploying off-the-shelf network switches using Clos topology and connecting each server to multiple other servers using multiple NICs. Scalability in DCNs is another challenge that needs to be tackled carefully.

DCNs in a disaggregated architecture needs to be considered carefully as disaggregation, i.e. resource pooling, can lead to higher latencies and an increase in bandwidth usage. To overcome both these issues, advanced optical interconnect technology and silicon photonics need to be deployed to replace standard electrical communication technology. Silicon photonics can remove the overhead caused by the use of several different components, such as optical waveguides, modulators, and photo-detectors, etc. in conventional electro-optical devices by integrating most of these functions into a single device. As discussed in a paper by IBM researchers, memory disaggregation can only be successful when ultra low latency links are available to connect disaggregated resources. [11] As disaggregation leads to higher latencies and bandwidth usage, the paper also introduced a simple cost model illustrated as follows,

$$G = MS - (CL + CB)$$

where G is the net gain expected from memory disaggregation, MS is the memory savings due to resource (memory) pooling, CL is the cost of increased latency, and CB is the cost of increased bandwidth. [11] An important conclusion drawn in the paper stated that potential benefits of memory disaggregation can only be achieved if it is implemented at the rack level to minimise distance; and also suggested that additional layers of cache memory, i.e. local memory acting as a cache to pooled remote memory, can aid in improving the performance of disaggregated architectures. [11]

[1] *RU* is an acronym for 'rack unit' – a standard unit of measure that describes the height of electronic equipment designed to mount in a 19-inch rack or a 23-inch rack. [12]

3 Simulator Framework

The simulator has been designed in MATLAB and uses its parallel execution feature (*parfor*) to improve and speed up the simulation process. Figure 12 shows the six major steps involved in the simulation process, each having a completely distinct role. The first step in the simulation involves reading multiple DCN configuration files that contain several configurable parameters; further discussion about these parameters is provided in the Configuration Files section. The second step involves generating multiple requests based on several predefined constraints. All generated requests are stored in a database, i.e. a request database for resource allocation and future reference. The next step involves setting up a parallel simulation environment using MATLAB's parallel computing toolbox. When parallelising, the process can simulate the three different types of data centre architectures created at the same time; these architectures are further discussed in the Data Centre Architecture section. The step following this is one in which all the resource maps and graphs are created based on the DCN configuration files read in the first step. These include connectivity, resource, bandwidth, and distance maps/graphs, etc. This is followed by the resource allocation step in which each request stored in the request database is scanned and allocated both IT and network resources. The last step in the simulation involves generating and displaying results and graphs that are used to analyse the performance of different data centre configurations and resource allocation algorithms used in the simulator. The following sections provide further implementation details for each step involved in the simulation process.

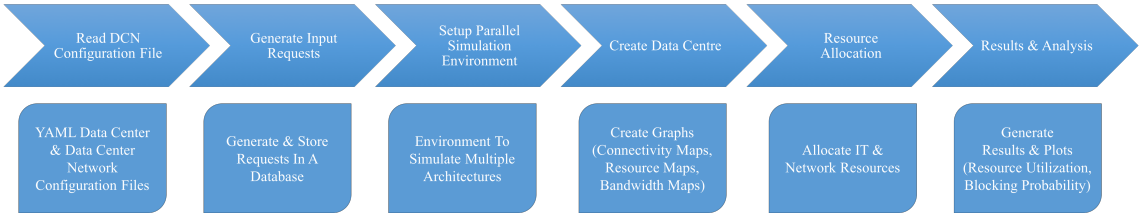


Figure 12: *High-level simulator flow chart illustrating the six major steps involved in the simulation process.*

3.1 Data Centre & Data Centre Network Generation

3.1.1 Configuration Files

The disaggregated data centres and data centre networks that are simulated are created based on several parameters specified in the configuration files. These configuration files use standard YAML syntax and are parsed by the simulator before setting up the parallel execution environment. Tables 1, 2, 3, 4, 5, 6, 7, 8, 9 show all the configurable parameters that are used by the simulator. Each of these parameters are completely configurable and can be changed to different values before starting the simulation.

Parameter	Description	Value
nRacks	Number of racks in the data centre	12
nBlades	Number of blades in a rack	16
nSlots	Number of slots in a blade	8
nUnits	Number of units in a slot	8
nTOR	Number of ToR switches in a rack	2
nTOB	Number of ToB switches in a blade	2

Table 1: *Configurable data centre parameters.*

Link	Description	Value
TOR_IntraRack	ToR-ToR link distance (Intra-rack)	0.50 m
TOR_InterRack	ToR-ToR link distance (Inter-rack)	2.00 m
TOR_TOB	ToR-ToB link distance	0.10 m
TOB_IntraBlade	ToB-ToB link distance (Intra-rack)	0.05 m
TOB_InterBlade	ToB-ToB link distance (Intra-rack)	0.10 m
TOB_slot	ToR-Slot link distance	0.05 m
slot	Slot-Slot link distance	0.05 m

Table 2: *Configurable link distance parameters.*

Link	Description	Value
TOR_TOR	Number of channels on a ToR-ToR link	8
TOR_TOB	Number of channels on a ToR-ToB link	8
TOB_TOB	Number of channels on a ToB-ToB link	1
TOB_slot	Number of channels on a ToB-Slot link	4
slot_slot	Number of channels on a Slot-Slot link	1

Table 3: *Configurable link channel parameters.*

Table 1 specifies the number of racks, blades, slots, top-of-rack (ToR), and top-of-blade (ToB) switches that are used in the data centre.

Table 2 specifies the the link distances between various combinations of nodes in the data centre. The values used here are based on the standard rack unit (RU) measurements

Hierarchy	Description	Value
rack	Rack-rack (ToR-ToR) topology	Fully-connected
rack_blade	Rack-blade (ToR-ToB) topology	Disconnected
blade	Blade-blade (ToB-ToB) topology	Spine-leaf
blade_slot	Blade-slot (ToB-Slot) topology	Spine-leaf
slot	Slot-Slot topology	Disconnected

Table 4: *Configurable network topology parameters.*

Resource Unit	Description	Value
CPU	Size of each compute unit	4 cores
MEM	Size of each memory unit	4 GBs
STO	Size of each storage unit	64 GBs

Table 5: *Configurable resource unit size parameters.*

Switch	Description	Value
TOD	Top of data centre (ToD) switch delay	10 ns
TOR	Top of rack (ToR) switch delay	10 ns
TOB	Top of blade (ToB) switch delay	10 ns

Table 6: *Configurable switch delay parameters.*

Constraint	Description	Value
minChannelLatency	Minimum channel latency	5 ns/m
maxChannelBandwidth	Maximum channel bandwidth	400 Gb/s
defaultDelay	Default in/out delay	200 ns

Table 7: *Configurable constraint (bound) parameters.*

– all values represent the standard 2U rack unit measurements in meters. These values represent distances between adjacent racks/blades/slots, whereas distances between non-adjacent racks/blades/slots have a multiplicative factor that is based on the difference between the source and destination racks/blades/slots.

Table 3 specifies the number of channels available on each type of link. Inter-switch

Setup Type	Description	Value
homogenCPU	Homogeneous compute blade	1
homogenMEM	Homogeneous memory blade	2
homogenSTO	Homogeneous storage blade	3
heterogenCPU_MEM	Heterogeneous compute/memory blade	4

Table 8: *Configurable blade type parameters.*

Parameter	Description	Value
racksConfig	Configuration of all racks	Various
heterogenSplit	Compute/memory percentage split	50

Table 9: *Other configurable parameters.*

links have been set up to contain a higher number of channels as compared to the other links in the data centre.

Table 4 specifies the network topology at each level in the network hierarchy. Several different network topologies are available for each level; these are further discussed in the Data Centre Network Architecture section.

Table 5 specifies the unit sizes for each resource type in the data centre. Slots are filled with multiple units, each having one of these predefined sizes.

Table 6 specifies the different switch delays in the data centre. Currently, all switches have the same delay but they can be configured to have different delays.

Table 7 specifies the minimum channel latency, maximum channel bandwidth, and the default in/out delay constraints.

Table 8 specifies the different values used for configuring the racks in the data centre. Each blade can be configured with only one of these values. Different combinations of these are used to create different types of disaggregated architectures.

Table 9 specifies the parameters used to configure all the racks in the data centre. The *racksConfig* parameter is a data structure containing *nRacks* arrays, i.e. cell arrays, each of size *nBlades* that are filled with values shown in Table 8. Further details on the architectures generated using different configurations are provided in the following section.

3.1.2 Data Centre Architecture

Using the parameters specified in the configurations files, different data centre and data centre network architectures can be created. The main three types of architectures created were **homogeneous racks with homogeneous blades**, **heterogeneous racks with homogeneous blades**, and **heterogeneous racks with heterogeneous blades**. These

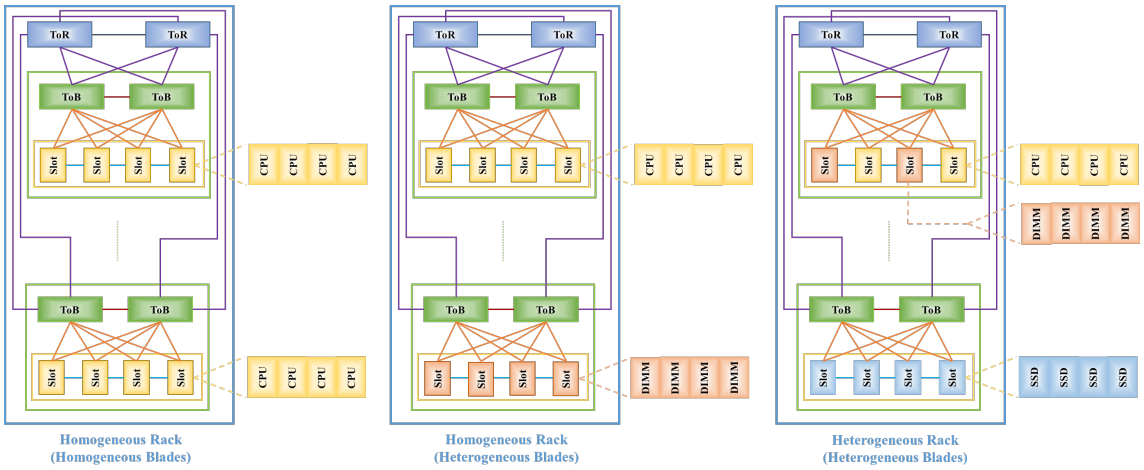


Figure 13: *The three different types of architectures that were used in the simulator. The rack on the left illustrates the architecture of a completely homogeneous rack; the rack in the centre illustrates the architecture of a heterogeneous rack with homogeneous blades; and the rack on the right illustrates the architecture of a heterogeneous rack with heterogeneous blades.*

architectures are illustrated in Figure 13. The rack on the left illustrates the architecture of a homogeneous rack with homogeneous blades containing only CPUs; the rack in the centre illustrates a heterogeneous rack with homogeneous blades containing both CPUs and memory in different blades; and the rack on the right illustrates a heterogeneous rack with heterogeneous blades containing both CPUs and memory in the same blades. The performance across these architectures can be extremely different depending on the resource allocation algorithm used; these are discussed in the Results section.

3.1.3 Data Centre Network Architecture

The simulator supports several network topologies for each level in the network hierarchy. Each of these topologies can be configured using the DCN configuration files parsed by the simulator. Figure 14 illustrates the graph based hierarchical structure of the data centre that is created by the simulator. In this graph, all terminal nodes represent slots, whereas all non-terminal nodes represent switches, i.e. either a ToR or a ToB switch. Inter-rack, inter-blade and inter-slot networks can be configured to contain either a fully-connected, line, ring or star topology, whereas networks that connect different levels in the hierarchy, such as rack-blade, i.e. ToR-ToB, and blade-slot, i.e. ToB-slot, networks can be configured to contain either a star or spine-leaf topology. These network topologies are shown in Figure 15. From a network perspective, all racks in the data centre; all blades within a rack; and all slots within a blade are self-contained. This means that connections between slots in different blades have to go through the top-of-blade (ToB) switches of both the source and destination slots; and connections between blades in different racks have to go through the top-of-rack (ToR) switches of both the source and destination blades, i.e. there are no direct connections between slots in different blades, and blades

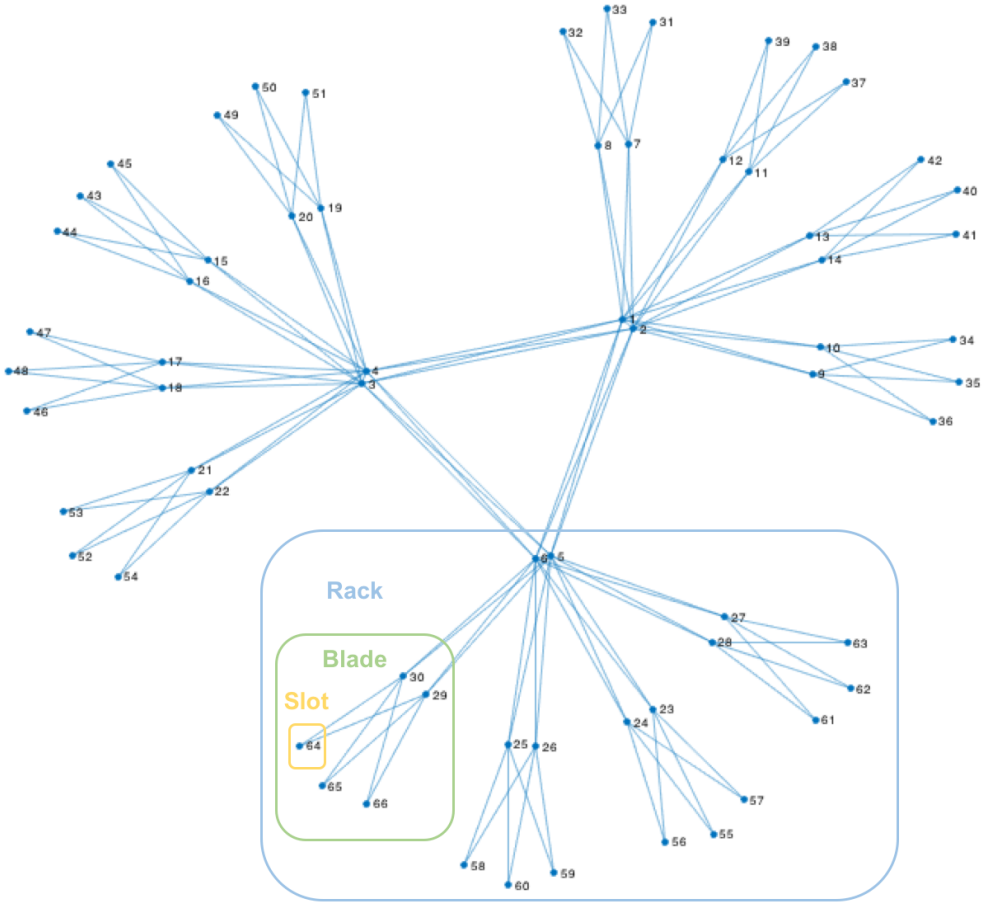


Figure 14: *An example connectivity graph illustrating the hierarchical structure created by the simulator; it represents a data centre with three racks, four blades in each rack, and three slots in each blade; each rack consists of two top-of-rack (ToR) switches and each blade consists of two top-of-blade switches. Each terminal node represents a slot whereas each non-terminal nodes represent a switch, either a ToR or ToB switch. Note that edge/link lengths are not to scale.*

in different racks. The number of top-of-rack (ToR) and top-of-blade (ToB) switches, and their delays can be configured using the DCN configuration files. Each circuit-switched connection, i.e. a physical channel between two slots, that is created during the allocation of network resources has a default in/out delay which has been set to 200 ns.

3.2 Input Request Generation

3.2.1 Request Constraints

All values generated for requests are based on a discrete uniform distribution with a minimum and a maximum bound. The minimum and maximum bounds for IT and network resources for a request are provided in Table 10. Figure 16 shows the distributions of values generated for all input request parameters. It can be seen that except for the memory parameter, all other parameters follow a discrete uniform distribution that lies

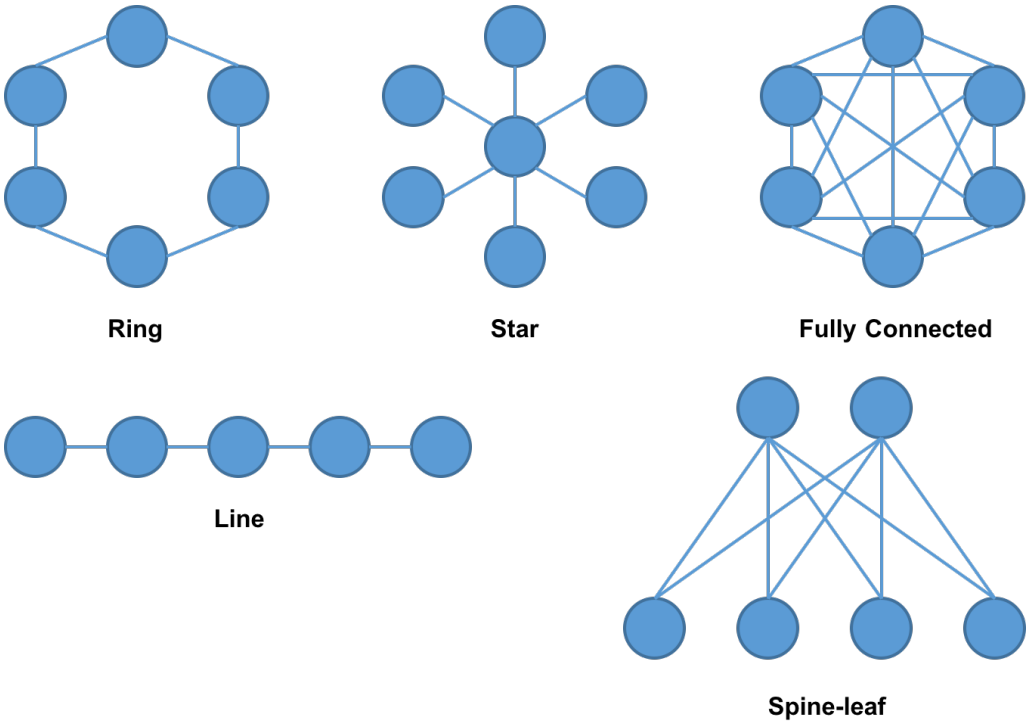


Figure 15: *Network topologies supported by the simulator.*

within the minimum and maximum bounds. For every request, the amount of memory requested is dependent on the number of CPUs requested; this is generated using a simple logarithmic formula that relates CPUs and memory requested. Due to the use of the logarithmic operation, there is a wide range of CPU values for which a large memory value is requested, therefore, making the demand for high memory values more likely. This is why the memory parameter does not completely follow a discrete uniform distribution.

Parameter	**Minimum**	**Maximum**
CPU	1 core	32 cores
Memory	1 GB	32 GBs
Storage	64 GBs	256 GBs
Bandwidth (CPU-Memory)	25 Gb/s	100 Gb/s
Bandwidth (Memory-Storage)	5 Gb/s	25 Gb/s
Latency (CPU-Memory)	300 ns	600 ns
Latency (Memory-Storage)	600 ns	1200 ns
Holding time	1 s	1000 s

Table 10: *Input/Request IT and network resource constraints/bounds.*

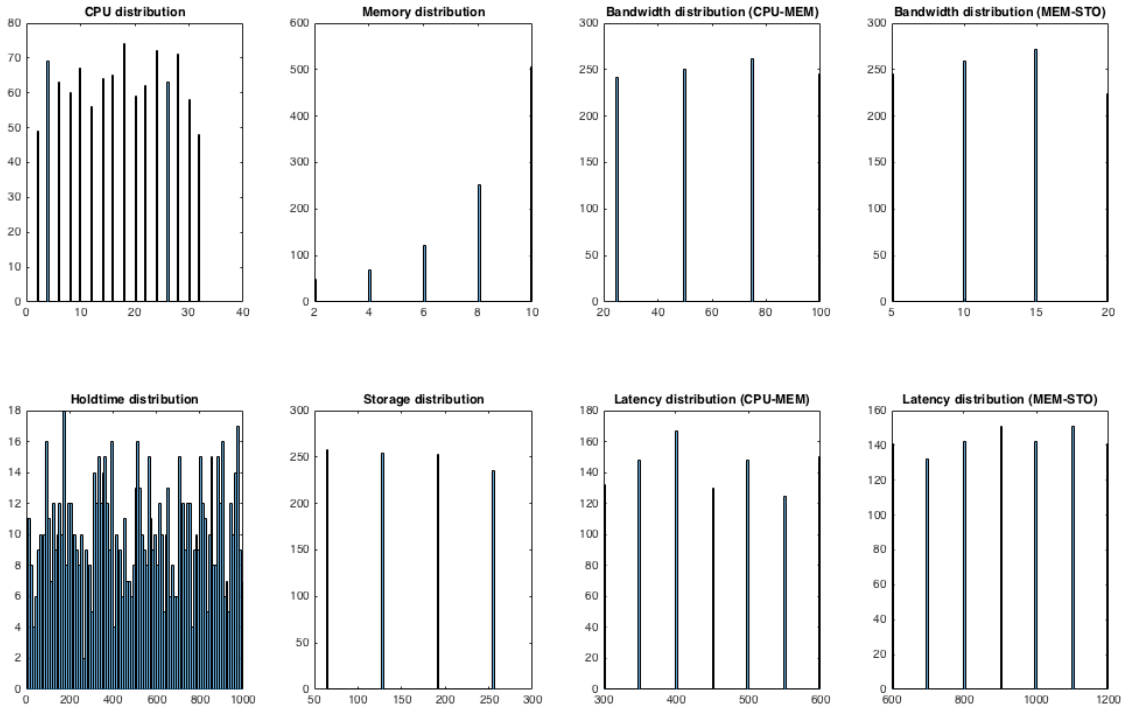


Figure 16: *Graphs illustrating the discrete uniform distributions (with minimum and maximum bounds) for each input request parameter.*

Network requirements for connections between CPU and memory units are a lot stricter in comparison to those between memory and storage units. The CPU-memory and memory-storage latency requirements differ by a factor of two, whereas their bandwidth requirements differ by a factor of five; this can be seen in Table 10.

3.2.2 Request Database

The request database is created to store all generated requests for resource allocation and future reference. For each request, it contains several fields that are populated and updated throughout the simulation. These fields include the IT resources (CPU, memory, and storage) requirements, network resource (bandwidth) requirements and latency constraints, holding time, request status and failure causes, etc. An example entry from the request database is shown in Table 11. At the end of the simulation, this database is also used to generate results as it keeps track of resource allocation status, failure cause, and allocation time, etc. for every request.

3.3 Resource Allocation

The resource allocation step is where requests get extracted from the request database and get their required IT and network resources allocated. Several algorithms were developed for IT and network resource allocation; these are discussed in the Resource Allocation Algorithms section. All algorithms that were developed perform several common micro-

Parameter/Constraint	Description	Value
CPU	Number of CPU units requested	32 cores
Memory	Number of memory units requested	10 GBs
Storage	Number of storage units requested	128 GBs
Bandwidth (CPU-Memory)	CPU-Memory bandwidth requested	75 Gb/s
Bandwidth (Memory-Storage)	Memory-Storage bandwidth requested	10 Gb/s
Latency (CPU-Memory)	CPU-Memory latency requested	600 ns
Latency (Memory-Storage)	Memory-Storage latency requested	1000 ns
Holding time	Resource holding time requested	475 s
Arrival time	Arrival time of request	220 s
IT allocation status	IT resource allocation status	Success
Network allocation status	Network resource allocation status	Success
Request status	Request status	Success
IT resources allocated	IT allocated node references	Various
Network resources allocated	Network link references	Various
Path latencies	Latencies of paths routed	Various
IT failure cause	IT allocation failure case	None
Network failure cause	Network allocation failure case	None
Allocation time	Time taken to find and allocate resources	2.36 s

Table 11: *An example entry extracted from the request database containing several fields such as the IT and network resource requirements, holding time, IT and network resource allocation statuses, etc.*

steps, such as extracting the request from the database, scanning for resources, allocating resources, and updating several graphs/maps that track IT and network resource utilisation, etc.

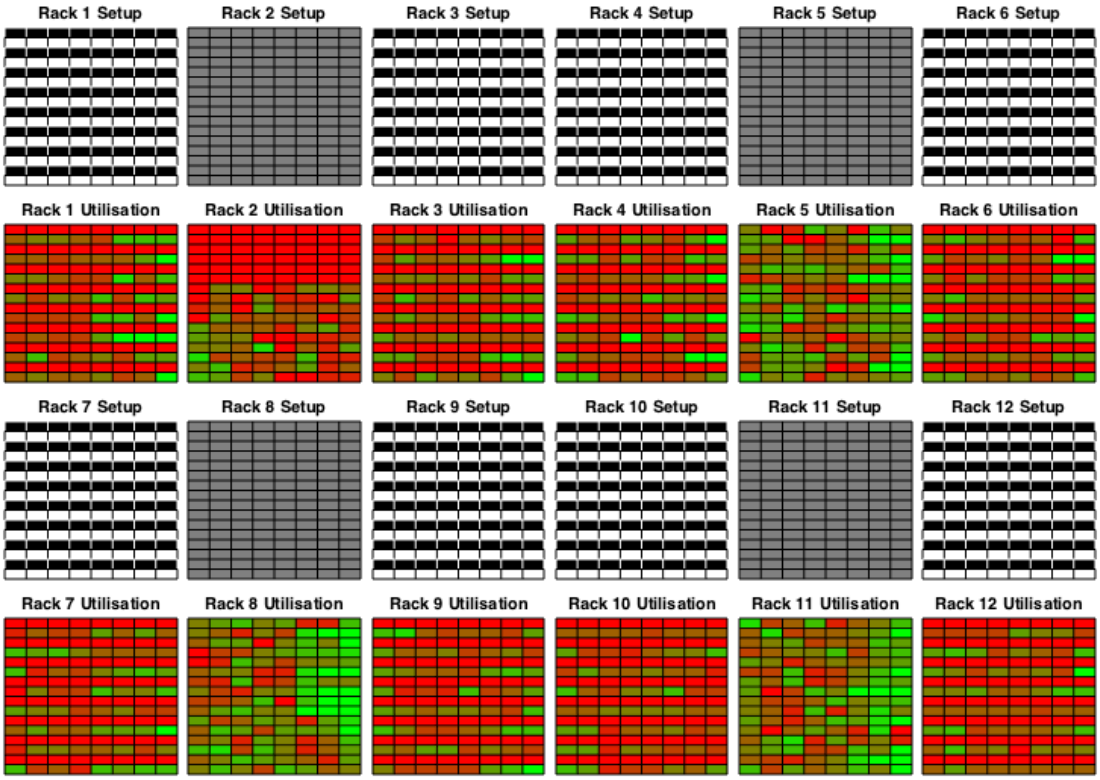


Figure 17: *Example heat map for a data centre containing heterogeneous racks with homogeneous blades. Each large rectangle represents a rack; each row within this large rectangle represents a blade and each column represents a slot within a blade. Black, white and grey rectangles represent CPU, memory and storage slots respectively.*

3.4 Results & Analysis

This step involves generating and displaying results and graphs for all types of architectures simulated. These graphs represent various performance metrics such as blocking probability, IT and network resource utilisation, data centre heat maps, etc. The simulator also supports dynamic updates for heat maps; these updated heat maps can be visualised as each request gets it's required resources allocated. Discussions of all results for all different architectures simulated are provided in the Results section. An example heat map of a data centre containing heterogeneous racks with homogeneous blades is illustrated in Figure 17.

4 Resource Allocation Algorithms

Resource allocation is a process of assigning and managing resources. In terms of the algorithms developed, these are IT and network resources that are required to successfully serve a request. Since developing a globally optimal solution to such a problem is incredibly difficult, the application of heuristics that are locally optimal, and those that take a strategic approach to resource allocation can yield higher utilisation and success rates. As discussed previously, disaggregation in particular has a huge impact on the latency and bandwidth available/required on networks connecting different resources as these resources can be placed far apart (possibly in different racks). Therefore, the algorithms need to carefully consider these factors when allocating network resources in disaggregated architectures. Different resource allocation algorithms can perform differently on

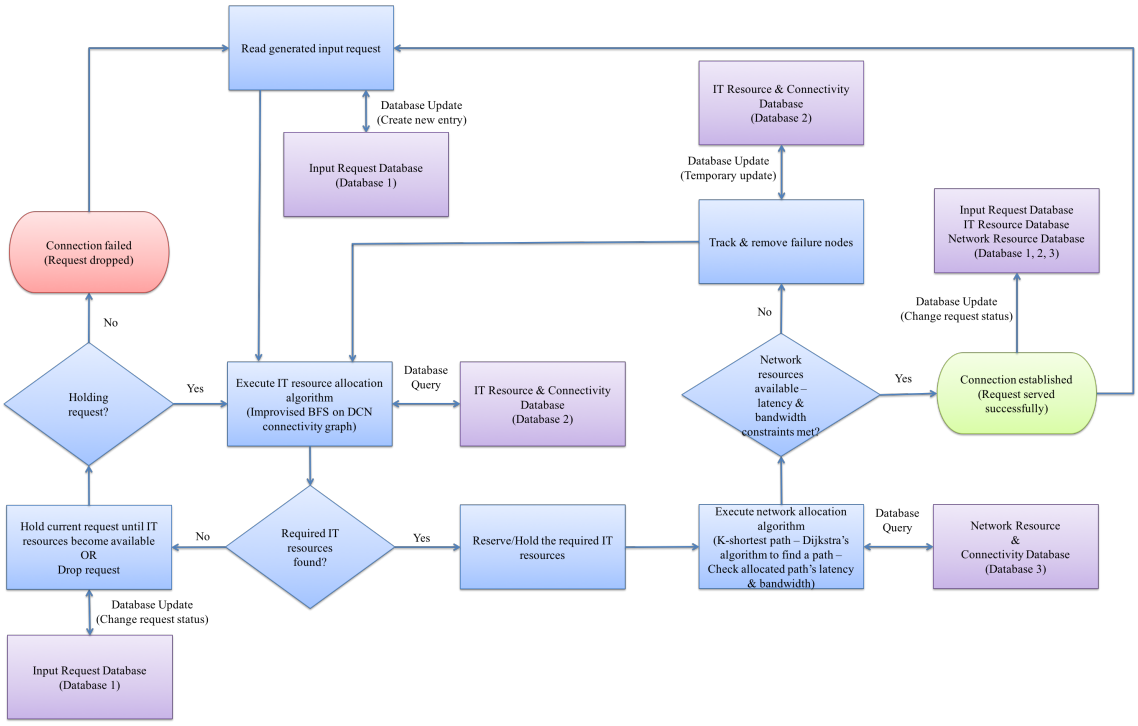


Figure 18: *A simple algorithmic flow-chart illustrating the steps involved in the resource allocation process.*

different disaggregated architectures; hence, to reduce/avoid this difference in performance across architectures and to keep the algorithms independent of the architectures they are run/used on, appropriate measures were taken when designing these algorithms. For every algorithm the standard flow is the same; in chronological order, it involves generating and reading a request, allocating IT resources, and allocating network resources including updating several databases where necessary. These databases include the request database; IT resource and connectivity database; and network resource database.

Algorithm 1: Simulator script common to all algorithms.

```
1: script SIMULATOR
2:     config ← Read YAML configuration files
3:     requestDatabase ← Generate input request database
4:     Setup parallel simulation environment
5:     for thread t in threadPool do
6:         switch t do
7:             case 1                                          ▷ Thread 1 (Type 1)
8:                 Initialise and start simulation
9:                 resultsDatabaseT1 ← SIMSTART(config, requestDatabase, t)
10:            case 2                                          ▷ Thread 2 (Type 2)
11:                Initialise and start simulation
12:                resultsDatabaseT2 ← SIMSTART(config, requestDatabase, t)
13:            case 3                                          ▷ Thread 3 (Type 3)
14:                Initialise and start simulation
15:                resultsDatabaseT3 ← SIMSTART(config, requestDatabase, t)
16:        end switch
17:    end for
18:    Generate and display results and graphs for all types
19: end script
```

The pseudocode for the script used to initialise, run and manage the entire simulation is provided in Algorithm 1 and the pseudocode for the function used to initialise the simulation for each type of disaggregated architecture is provided in Algorithm 2.

Algorithm 2: Simulator function common to all algorithms.

```
Require: config: Configuration, requestDatabase: Request database, t: Thread ID
1: procedure SIMSTART(config, requestDatabase, t)
2:     G ← Initialise and create data centre graphs
3:     for each request r in requestDatabase do
4:         ▷ Allocate IT and network resources for request r
5:         allocationResult ← RESOURCEALLOCATION(config, G, r)
6:         Update requestDatabase
7:     end for
8:     return requestDatabase
9: end procedure
```

4.1 Formal Definition

4.1.1 Bin packing problem

In computational complexity theory, the standard bin packing problem is a combinatorial NP-hard problem (whereas deciding the optimal number of bins is a NP-complete problem). [13] It requires objects of different volumes to be packed into a finite number of bins

or containers each of volume V in a way that minimises the number of bins used. Different variations of of bin packing exist and can be applied in multiple dimensions. Despite the bin packing problem being classed as a NP-hard problem, several optimal solutions can be produced with the use of sophisticated algorithms. [13]

Formalisation

Given a bin S of size V and a list of items with sizes $a_1, \dots, a_n$ to pack, the problem is to find an integer number B and a B – partition $S_1 \cup \dots \cup S_B$ of the set $\{1, \dots, n\}$ such that $\sum_{i \in S_k} a_i \leqslant V \ \forall k = 1, \dots, B$. A solution is said of be optimal if it has minimal B. [13] In simple terms, the problem is to find the minimal number of bins in which all items in the list can be packed. An integer linear optimisation model of problem is as follows: [14]

$$
\begin{aligned}
\text{minimise} \quad & B = \sum_{i=1}^{n} y_i \\
\text{subject to} \quad & B \geqslant 1, \\
& \sum_{j=1}^{n} a_j x_{ij} \leqslant V y_i, && \forall i \in \{1, \dots, n\} \\
& \sum_{i=1}^{n} x_{ij} = 1, && \forall j \in \{1, \dots, n\} \\
& y_i \in \{0, 1\}, && \forall i \in \{1, \dots, n\} \\
& x_{ij} \in \{0, 1\}, && \forall i \in \{1, \dots, n\} \ \forall j \in \{1, \dots, n\} \\
\text{where} \quad & y_i = \begin{cases} 1 & \text{if bin } i \text{ is used} \\ 0 & \text{otherwise} \end{cases} \\
& x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is assigned to bin } i \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

The problem that the resource allocation algorithms developed try to solve can be reduced to the bin-packing problem that is implemented on a connected graph. While standard algorithms for the bin packing problem try to find an optimal (minimal) number of bins in which all items can be packed, the resource allocation algorithms developed do the inverse – given a predefined (set) number of bins, pack as many items as possible. The items here refer to IT (CPU, memory, and storage) and network resources across all requests. This can also be visualised as a 2-D bin packing problem, where in the first dimension, the algorithms try to maximise the IT resources allocated, and in the second dimension, they try to minimise the network resources allocated while satisfying the constraints – these constraints are the latency and bandwidth requirements of a request. Another way to look at the problem would be to say serve as many requests as possible without saturating the data centre network.

There are two major versions of algorithms, online and offline algorithms. In the

online version only a single request is known at one time, whereas in the offline version all requests are known upfront; the resource allocation algorithms developed implement the online version. It is obvious that with the online version always finding an optimal solution is difficult as future requests are unknown but with a heuristic approach and approximations the algorithms developed can achieve a near optimal solution.

4.1.2 Problem Formalisation

In terms of the algorithms developed, given requests $r_1, \dots, r_m$, where only a single request r_i is known at a point in time, a list of required IT resources $a_1, \dots, a_q$, and a list of required connections $c_1, \dots, c_t$, each containing edges $e_1, \dots, e_n$, assigned to connect all required IT resources for a particular request; the problem is to minimise the total allocated bandwidth B across the entire data centre network while satisfying every request's latency and bandwidth constraints; and maximise the total allocated IT resources S across all requests. Similar to the linear optimisation model for the bin packing problem, one for the resource allocation problem is as follows:

$$
\begin{array}{lllll}
\text{minimise} & B = \sum_{k=1}^{m} \sum_{i=1}^{t} c_{ik} e_{ij}, & & & \forall j \in \{1, \dots, n\} \\
\text{subject to} & e_{ij}^{b} \geqslant r_{k}^{b}, & \forall i \in \{1, \dots, t\} & \forall j \in \{1, \dots, n\} & \forall k \in \{1, \dots, m\} \\
& B \leqslant B' & & & \\
& c_{i}^{l} \leqslant r_{k}^{l}, & & \forall i \in \{1, \dots, t\} & \forall k \in \{1, \dots, m\} \\
\text{maximise} & S = \sum_{k=1}^{m} a_{ik}, & & & \forall i \in \{1, \dots, q\} \\
\text{subject to} & S \leqslant S' & & &
\end{array}
$$

where

r^b is the required bandwidth for request r

r^l is the required latency for request r

e^b is the bandwidth allocated on edge e

c^l is the latency on connection c

B' is the total network bandwidth

S' is the total IT resources

$$
c_{ik} = \begin{cases} 1 & \text{if connection } i \text{ is assigned to request } k \\ 0 & \text{otherwise} \end{cases}
$$

$$
e_{ij} = \begin{cases} r^b & \text{if edge } j \text{ is allocated in connection } i \\ 0 & \text{otherwise} \end{cases}
$$

$$
a_{ik} = \begin{cases} 1 & \text{if resource } i \text{ is allocated to request } k \\ 0 & \text{otherwise} \end{cases}
$$

Let $G = (V, E)$ be an undirected connected graph that represents the entire data centre, where, V is the set of all vertices present in the data centre, and E is the set of edges, each of which is a set of two vertices. All vertices in the graph represent either an IT resource slot or a switch and all edges in the graph represent a network link present in the data centre network.

Let $n = |V|$ be the total number of vertices in the graph, and $m = |E|$ be the total number of edges in the graph.

Let v^i represent the i^{th} vertex in the set of vertices V and e^i represent the i^{th} edge in the set of edges E. Using these denotations, we get the following:

$$V_i = \{v^i \ \forall i \mid i \in \mathbb{N}, 1 \leqslant i \leqslant n\}$$
$$E_i = \{e^i \ \forall i \mid i \in \mathbb{N}, 1 \leqslant i \leqslant m\}$$

Instead of using an unique identifier for an edge, an alternative representation would be to use superscripts that specify its source and destination vertices, for example, $e^{i,j}$ would represent an edge between vertices v^i and v^j; but this notation is avoided for simplicity.

All vertices and edges have a minimum and maximum capacity bound. For vertices, it is the amount of IT resource units (CPU, memory or storage) present at a vertex, and the capacity on the i^{th} vertex is represented by v^i_c. For edges, it is the amount of bandwidth available on an edge, and the capacity on the i^{th} edge is represented by e^i_c. Let $v^i_{c,min}$ and $v^i_{c,max}$ be the minimum and maximum capacities of a vertex v^i; and $e^i_{c,min}$ and $e^i_{c,max}$ be the minimum and maximum capacities of an edge e^i. We then define the sets of capacities as follows:

$$V_c = \{v^i_c \in \mathbb{N} \mid v^i_{c,min} \leqslant v^i_c \leqslant v^i_{c,max}, i \in \mathbb{N}, 1 \leqslant i \leqslant n\}$$
$$E_c = \{e^i_c \in \mathbb{N} \mid e^i_{c,min} \leqslant e^i_c \leqslant e^i_{c,max}, i \in \mathbb{N}, 1 \leqslant i \leqslant m\}$$

Each edge also has a distance associated with it, which defines its latency. Let e^i_d be the distance of the i^{th} edge e^i and e^i_l be its latency. For the linear relation between an edge's distance and its latency, we define a new variable f. It follows that the latency e^i_l on an edge e^i is $f \times e^i_d$. We then define the set of distances and latencies as follows:

$$E_d = \{e^i_d \in \mathbb{R} \mid e^i_d > 0, i \in \mathbb{N}, 1 \leqslant i \leqslant m\}$$
$$E_l = \{e^i_l \in \mathbb{R} \mid e^i_l > 0, e^i_l = f \times e^i_d, f \in \mathbb{R}, i \in \mathbb{N}, 1 \leqslant i \leqslant m\}$$

For a request r, let r_c be the number of CPU units required, r_m be the number of memory units required, r_s be the number of storage units required, $r_{b,cm}$ be the CPU-memory bandwidth required, $r_{b,ms}$ be the memory-storage bandwidth required, $r_{l,cm}$ be the maximum acceptable CPU-memory latency, and $r_{l,ms}$ be the maximum acceptable memory-storage latency.

For every request, a new weighted graph is created $G' = (V, E')$ where E' is a new set

of edges where every edge is weighted on both its latency E_l and bandwidth E_c. Let this weighting factor be f'. We then define the set of new weighted edges E'_c as follows:

$$E'_c = \left\{ e'^i_c \in \mathbb{R} \mid e'^i_c = f' \times \left(1 - \frac{e''^i_c}{e^i_c}\right) + (1 - f') \times \frac{e^i_l}{e_{l,max}}, i \in \mathbb{N}, i \leqslant m, f' \in \mathbb{R}, f' \leqslant 1 \right\}$$

where e'^i_c represents the newly weighted i^{th} edge, e''^i_c represents its updated capacity, i.e. unused/remaining capacity, e^i_c represents its original capacity, e^i_l represents its latency and $e_{l,max}$ represents the maximum latency in the network. It is important to note that every i^{th} edge contained in E'_c satisfies the minimum required bandwidth for a request, $e'^i_c \geqslant min(r_{b,cm}, r_{b,ms})$. The new weighted graph is used for allocating network resources for a request; without this weighting, i.e. only considering either bandwidth or latency, there is a high probability of the request getting blocked/dropped since either constraints might not be satisfied when being checked one after the other.

To successfully allocate the resources required by a request r, the algorithm chooses and allocates sets of CPU, memory, and storage vertices that are all connected to each other and satisfy the required bandwidth and latency constraints. A more detailed formalisation could be made to represent different sets for different types of resources allocated including the link bandwidth and latency connecting these resources; providing these specific details formally can make the problem appear more complicated, hence, it is being avoided. Informally, the latency on connections between every CPU vertex to every other CPU and memory vertices $l_c \leqslant r^l_{cm}$; and the latency on all other connections $l_o \leqslant r^l_{ms}$. The bandwidth on each edge on paths connecting these allocated CPU vertices to other CPU and memory vertices $b_c \geqslant r^b_{cm}$; and the bandwidth on edges on paths connecting all other resources $b_o \geqslant r^b_{ms}$. After a request is successfully allocated, the sets of vertex capacities V_c and edge capacities E_c are updated ensuring that each element $v^i_c \geqslant v^i_{c,min}$ in V_c and $e^i_c \geqslant e^i_{c,min}$ in E_c respectively.

4.2 Graph Theory Algorithms

4.2.1 Breadth-first Search (BFS)

Breadth-first search (BFS) is an algorithm for traversing and searching tree and graph data structures. It traverses a graphs starting at an arbitrary vertex and searches all it's neighbouring vertices before moving to the next level of vertices. The algorithm uses a queue to store all discovered vertices; and all queued vertices get dequeued in a first-in first-out (FIFO) manner. The pseudocode for the BFS algorithm is provided in Algorithm 3.

Algorithm 3: Breadth-first search (BFS) algorithm

Require: G: Input graph, s: Start vertex

```
1: procedure BFS(G, s)
2:     for each vertex v in G do                ▷ Initialise each vertex
3:         v.distance ← ∞
```

```
4:        v.parent ← null
5:    end for
6:    Q ← [ ]                                          ▷ Create an empty queue
7:    s.distance ← 0
8:    Q.enqueue(s)                                     ▷ Enqueue start vertex
9:    while Q ≠ [ ] do                                 ▷ Run until queue is empty
10:       current ← Q.dequeue()
11:       for each vertex v adjacent to current do
12:           if v.distance = ∞ then
13:               v.distance ← current.distance + 1
14:               v.parent ← current
15:               Q.enqueue(v)                         ▷ Enqueue v^th neighbour
16:           end if
17:       end for
18:   end while
19: end procedure
```

The worst case time complexity of BFS is $O(|V|+|E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges in the input graph. Note that $O(E)$ can vary between $O(1)$ and $O(|V|^2)$ depending on the sparsity of the input graph. The BFS algorithm is used to find and allocate IT resources in both the locality based algorithms discussed in the Network-Unaware Locality Based Resource Allocation and Network-Aware Locality Based Resource Allocation sections.

4.2.2 K Shortest Path – Yen's Algorithm

The K shortest path algorithm is a generalisation and an extension to standard shortest path algorithms, where instead of just a single shortest path between two vertices, k shortest paths are found. To find the shortest path between two vertices, standard shortest path algorithms such as Dijkstra's algorithm or Bellman Ford algorithm can be used and are extended to find more than one path. Several different implementations for the K shortest path algorithm exist – the one used in the simulator is called the Yen's Algorithm. The pseudocode for Yen's algorithm is provided in Algorithm 4.

Algorithm 4: Yen's K shortest path algorithm

Require: G: Input graph, u: Source vertex, v: Destination vertex, K: K-shortest paths

```
1:  procedure YenKSP(G, u, v, K)
2:      A[0] ← Dijkstra(G, u, v)          ▷ Determine shortest path from u to v
3:      B[0] ← [ ]                                   ▷ Initialise empty heap
4:      for k from 1 to K do
5:          for i from 0 to size(A[k − 1]) − 1 do
6:              spurVertex ← A[k − 1].vertex(i)      ▷ Retrieved from k − 1 shortest path
7:              rootPath ← A[k − 1].vertices(0, i)   ▷ Path from source to spur vertex
8:              for each path p in A do
9:                  if rootPath = p.vertices(0, i) then
10:                     Remove p.edge(i, i + 1) from G
```

```
11:             end if
12:         end for
13:         for each vertex rootPathVertex in rootPath except spurVertex do
14:             Remove rootPathVertex from G
15:         end for
16:         spurPath ← DIJKSTRA(G, spurVertex, v)
17:         totalPath ← rootPath + spurPath
18:         B.append(totalPath)
19:         Restore edges to G                          ▷ Add edges removed from graph
20:         Restore vertices in rootPath to G           ▷ Add vertices removed from graph
21:         if B = [ ] then                  ▷ Handle case when there are no spur paths
22:             break
23:         end if
24:         B.sort()                          ▷ Sort potential k-shortest paths by cost
25:         A[k] ← B[0]                              ▷ Store k^th path with lowest cost
26:         B.pop()
27:     end for
28: end for
29: return A                                                ▷ Return k shortest paths
30: end procedure
```

Yen's algorithm computes single-source K-shortest loopless paths for a graph with non-negatively weighted edges. The K shortest path algorithm is used to find/route paths between allocated IT resource vertices that have an acceptable latency and to allocate network resources, i.e. bandwidth, on every path. This implementation of K shortest paths is used in all the resource allocation algorithms developed and is discussed in more detail in the sections ahead.

The implementation of Yen's algorithm in the simulator uses the Dijkstra's algorithm to find the single source shortest path. The pseudocode for Dijkstra's algorithm is provided in Algorithm 5. The runtime complexity of Yen's algorithm highly depends on the shortest path algorithm used; and its worst case time complexity when using Dijkstra's algorithm is $O(K|V|(|E| + |V|\log|V|))$, where $|V|$ represents the number of vertices, $|E|$ represents the number of edges in the graph, and K represents the number of shortest paths to be found.

Dijkstra's algorithm is used to find the shortest paths from the source vertex to all other vertices in the graph. The algorithm can be modified to accept a destination vertex; where every extracted vertex on line 12 (in Algorithm 5) would need to be checked to match the destination vertex, and if a match is found, it could break out of the main loop on line 11. Following this, a path from source to destination could be extracted by reversing the order of predecessor vertices. When using a minimum-priority queue implemented by a Fibonacci heap, the worst-case time complexity of Dijkstra's algorithm is $O(|E| + |V|\log|V|))$, where $|V|$ is the number of vertices in the graph and $|E|$ is the number of edges in the graph.

Algorithm 5: Dijkstra's algorithm

Require: G: Input graph, s: Start vertex, d: Destination vertex (Optional)

```
1:  procedure DIJKSTRA(G, s, d)
2:      Q ← [ ]                                          ▷ Create vertex set
3:      dist[s] ← 0
4:      for each vertex v in G do                        ▷ Initialise each vertex
5:          if v ≠ s then
6:              dist[v] ← ∞
7:              prev[v] ← null
8:              Q.addWithPriority(v, dist[v])            ▷ Add all vertices to Q
9:          end if
10:     end for
11:     while Q ≠ [ ] do                                 ▷ Run until queue is empty
12:         u ← Q.extractMin()                           ▷ Extract vertex with minimum distance
13:         for each vertex v adjacent to u do           ▷ Where v is still in Q
14:             alt ← dist[u] + length(u, v)
15:             if alt < dist[v] then                    ▷ Relax
16:                 dist[v] ← alt
17:                 prev[v] ← u
18:                 Q.decreasePriority(v, alt)
19:             end if
20:         end for
21:     end while
22:     return dist[ ], prev[ ]                          ▷ Return distances and predecessors
23: end procedure
```

4.3 First Fit Resource Allocation

The first fit resource allocation algorithm is the simplest of algorithms developed. In this algorithm resources are allocated on a first-fit basis. This means that every request is allocated the first available (IT) resource nodes found. Although this algorithm is simple, finding optimal solutions for a large number of requests is difficult. When looking for IT resource nodes, the algorithm does not consider the availability of network resources, therefore, there is a possibility of it choosing nodes that have IT resources available but links connecting these nodes are either completely saturated or do not satisfy the minimum bandwidth requirement for a request. This can result in a high blocking probability, i.e. a high request drop rate, particularly due to the algorithm failing when allocating network resources. For the allocation of network resources, the optimised network allocation algorithm is used in conjunction with **all** IT resource allocation algorithms. Even when using the most optimised network allocation algorithm, choosing and allocating IT resource nodes without considering the network resources can lead to poor performance. The network-aware locality based algorithm, discussed in the Network-Aware Locality Based Resource Allocation section, exploits this; and amongst other performance improvements, it is seen to have a significant drop in blocking probability.

Algorithm 6: First fit resource allocation algorithm

Require: G: Data centre graphs, C: Data centre config, R: Request

```
1:  procedure FIRSTFIT(G, C, R)
2:      available.Cpu ← find slots ⩾ 1 in C.CpuLocations
3:      available.Mem ← find slots ⩾ 1 in C.MemLocations
4:      available.Sto ← find slots ⩾ 1 in C.StoLocations
5:      if available.Cpu < R.Cpu then                    ▷ Inadequate cpu resources
6:          ITallocation ← Failure
7:          ITfailureCause ← Cpu
8:      else if available.Mem < R.Mem then               ▷ Inadequate memory resources
9:          ITallocation ← Failure
10:         ITfailureCause ← Mem
11:     else if available.Sto < R.Sto then               ▷ Inadequate storage resources
12:         ITallocation ← Failure
13:         ITfailureCause ← Sto
14:     else
15:         loopIncrement ← C.nSlots × C.nBlades         ▷ Size of a rack
16:         slot ← startSlot
17:         while slot ⩽ C.totalSlots do                 ▷ To try multiple combinations
18:             found.Cpu ← 0                            ▷ Initialise resource counter variables
19:             found.Mem ← 0
20:             found.Sto ← 0
21:             index.Cpu ← 1                            ▷ Initialise resource index variables
22:             index.Mem ← 1
23:             index.Sto ← 1
24:             ITres ← [ ]                              ▷ Initialise IT resource 'tracker'
25:             for each slot s in C.ITres do
26:                 switch (C.ITresTypes(s)) do          ▷ Different resource types
27:                     case Cpu                         ▷ Find cpu slots
28:                         if found.Cpu < R.Cpu then
29:                             units ← C.ITres(s)
30:                             found.Cpu ← found.Cpu + units
31:                             ITres[1, index.Cpu] ← {s, units}
32:                             index.Cpu ← index.Cpu + 1
33:                         end if
34:                     case Mem                         ▷ Find memory slots
35:                         if found.Mem < R.Mem then
36:                             units ← C.ITres(s)
37:                             found.Mem ← found.Mem + units
38:                             ITres[2, index.Mem] ← {s, units}
39:                             index.Mem ← index.Mem + 1
40:                         end if
41:                     case Sto                         ▷ Find storage slots
42:                         if found.Sto < R.Sto then
43:                             units ← C.ITres(s)
44:                             found.Sto ← found.Sto + units
45:                             ITres[3, index.Sto] ← {s, units}
46:                             index.Sto ← index.Sto + 1
```

```
47:                     end if
48:                 end switch
49:             end for
50:             if R.Cpu and R.Mem and R.Sto found then          ▷ All resources found
51:                 ITallocation ← Success
52:                 ITfailureCause ← None
53:                 break
54:             else
55:                 ITallocation ← Failure
56:                 Evaluate failure cause
57:                 Update ITfailureCause
58:             end if
59:             if ITallocation = Success then
60:                 [NETallocation, NETres] ← netAllocation(G, C, R, ITres)
61:                 if NETallocation = Success then
62:                     break
63:                 else
64:                     Remove failure nodes in C.ITres          ▷ Remove failure nodes
65:                 end if
66:             else
67:                 break                  ▷ Failed due to unavailability of IT resources
68:             end if
69:             slot ← slot + loopIncrement                      ▷ Jump to next rack
70:         end while
71:     end if
72:     if ITallocation = Success and NETallocation = Success then
73:         Update C.ITres                          ▷ Remove allocated IT resources
74:         Update G.bMap                    ▷ Remove allocated network resources
75:     end if
76:     return ITallocation, ITres, NETallocation, NETres
77: end procedure
```

The pseudocode for the first fit algorithm is provided in Algorithm 6. All instances of *Cpu*, *Mem*, and *Sto* refer to CPU, memory and storage respectively; and all slots refer to terminal nodes/vertices in the data centre. Note that all pseudocodes provided in this thesis only contain major (logic) sections of the algorithms and their actual implementations in a specific language require several other factors to be considered. In summary, the first fit resource allocation algorithm performs well early on when both the IT and network resources have a low utilisation, but as these resources start to saturate, the blocking probability for subsequent requests increases significantly. The latencies on network paths allocated by this algorithm can be worse as compared to those allocated by the other algorithms. Detailed results on its performance are discussed in the Results section.

Algorithm 7: Optimised network resource allocation algorithm

Require: G: Data centre graphs, C: Data centre config, R: Request, N: Allocated nodes

```
1: procedure NETALLOCATION(G, C, R, N)
2:     bMap ← G.bMap                                   ▷ Copy original bandwidth map
3:     dMap ← G.dMap                                   ▷ Copy original distance map
4:     Remove all edges in bMap and dMap where e_b < min(R.b_cm, R.b_ms)
5:     maxD ← maximum distance in dMap
6:     f ← 0.5                                         ▷ Weightage factor
7:     wGraph ← ∞                                      ▷ Initialise new weighted graph
8:     NETres ← [ ]                                    ▷ Initialise network resource 'tracker'
9:     for each edge e in bMap do
10:        if dMap(e) ≠ ∞ and bMap(e) > 0 then
11:            wGraph(e) ← f × (1 − bMap(e)/G.bMap(e)) + (1 − f)(G.dMap(e)/maxD)
12:        end if
13:    end for
14:    K ← 3                                           ▷ K-shortest paths to find
15:    for each node u in N do
16:        for each node v in N do
17:            paths ← YENKSP(wGraph, u, v, K)
18:            Find link latency on every path
19:            Find switch latency on every path
20:            L(u, v, k) ← lLat + sLat + dLat         ▷ Total latency on k^th path
21:        end for
22:    end for
23:    for each node u in N do
24:        for each node v in N do
25:            for each path k in paths do
26:                if L(u, v, k) ⩽ R.l_cm or R.l_ms then   ▷ Context dependent
27:                    L'(u, v, k) ← 1                 ▷ Acceptable latency path
28:                else
29:                    L'(u, v, k) ← 0                 ▷ Unacceptable latency path
30:                end if
31:            end for
32:        end for
33:    end for
34:    for each node u in N do
35:        for each node v in N do
36:            for each path k in paths do
37:                Check L' to see if at least a single acceptable path has been found
38:                Track failure nodes in failureNodes
39:                Update acceptableLatencyPathsFound
40:            end for
41:        end for
42:    end for
43:    if acceptableLatencyPathsFound = True then
44:        latency ← Success
45:    else
46:        latency ← Failure                           ▷ Failed due to latency
47:        NETfailureCause ← Latency
48:    end if
49:    if latency = Success then
```

```
50:         for each node u in N do
51:             for each node v in N do
52:                 bMapRevert ← bMap                    ▷ Copy to revert if k^th path fails
53:                 for each path k in paths do
54:                     if L'(u, v, k) = 1 then
55:                         for each edge e in path k do
56:                             if bMap(e) ⩾ R.b_cm or R.b_ms then      ▷ Context dependent
57:                                 Allocate and update bandwidth on bMap(e)
58:                             else
59:                                 Track failure nodes in failureNodes
60:                                 break
61:                             end if
62:                         end for
63:                         if bandwidth allocation on path k is successful then
64:                             Track all successful paths in NETres
65:                             Track latency on all allocated paths
66:                             Update acceptableBandwidthPathsFound
67:                         else
68:                             bMap ← bMapRevert                ▷ Since k^th path failed
69:                         end if
70:                     end if
71:                 end for
72:             end for
73:         end for
74:     end if
75:     if acceptableBandwidthPathsFound = True then
76:         bandwidth ← Success
77:     else
78:         bandwidth ← Failure                          ▷ Failed due to bandwidth
79:         NETfailureCause ← Bandwidth
80:     end if
81:     if latency = Success and bandwidth = Success then
82:         NETallocation ← Success                      ▷ Network allocation successful
83:         NETfailureCause ← None
84:         G.bMap ← bMap                                ▷ Update original bandwidth map
85:     else
86:         NETallocation ← Failure                      ▷ Need to find new set of IT resources
87:     end if
88:     return NETallocation, failureNodes, NETres
89: end procedure
```

The pseudocode for the optimised network allocation algorithm that is used in conjunction with all algorithms is provided in Algorithm 7. After the initialisation stage, a new weighted graph is created as shown on line 11 (in Algorithm 7) where every edge is weighted on both it's latency and available bandwidth based on the factor f. A factor value close to 1 favours bandwidth, whereas a factor value close to 0 favours latency. For the simulations both factor were weighted equally and the value for f was set to 0.5. The weighting is important to decrease the probability of a certain potential network path

being blocked when either factors are considered one after the other. Line 26 and 56, both have different latency and bandwidth checks for CPU-CPU, CPU-memory, CPU-storage, memory-memory, and memory storage depending on the constraints defined by a request. After allocating the bandwidth on each edge of a path $bMap$ is updated to reflect the new available bandwidth on that edge. If all k-paths, i.e. connections, between a pair of IT resource nodes fail, these failure nodes are tracked and returned; this is done to prevent the IT resource allocation algorithms from considering these nodes when it (scans for and) allocates a new set of IT resource nodes. The variables $lLat$, $sLat$, and $dLat$ on line 20 refer to link, i.e. edge, switch, and default in/out latencies. The value for K was set to 3 – this ensures that exactly three shortest paths are found between every allocated IT resource node.

4.4 Best Fit Resource Allocation

The best fit resource allocation algorithm is one in which the best possible combination of IT resources are allocated to every request. In comparison to the first fit algorithm, the major difference in its implementation is that different types of resources are stored and managed separately (using different data structures). The advantage of this is that the algorithm can jump to different racks for every resource type, whereas this is not possible in the first fit algorithm since all IT resources are managed using a single data structure. As mentioned previously, for allocating network resources, the same optimised network allocation algorithm is used (Refer to pseudocode provided in Algorithm 7). The best fit algorithm has a significantly better performance in all aspects, especially in terms of blocking probability – a lot more requests are successfully allocated their required resources in comparison to the first fit algorithm. The pseudocode for the best fit resource allocation algorithm is provided in Algorithm 8.

Algorithm 8: Best fit resource allocation algorithm

Require: G: Data centre graphs, C: Data centre config, R: Request

```
1:  procedure FIRSTFIT(G, C, R)
2:      available.Cpu ← find slots ⩾ 1 in C.CpuLocations
3:      available.Mem ← find slots ⩾ 1 in C.MemLocations
4:      available.Sto ← find slots ⩾ 1 in C.StoLocations
5:      if available.Cpu < R.Cpu then                    ▷ Inadequate cpu resources
6:          ITallocation ← Failure
7:          ITfailureCause ← Cpu
8:      else if available.Mem < R.Mem then               ▷ Inadequate memory resources
9:          ITallocation ← Failure
10:         ITfailureCause ← Mem
11:     else if available.Sto < R.Sto then               ▷ Inadequate storage resources
12:         ITallocation ← Failure
13:         ITfailureCause ← Sto
14:     else
15:         loopIncrement ← C.nSlots × C.nBlades         ▷ Size of a rack
```

```
16:     slotLimit ← max(C.CpuLocations, C.MemLocations, C.StoLocations)
17:     slot ← 1
18:     while slot < slotLimit do                          ▷ To try multiple combinations
19:         found.Cpu ← 0                                  ▷ Initialise resource counter variables
20:         found.Mem ← 0
21:         found.Sto ← 0
22:         index.Cpu ← 1                                  ▷ Initialise resource index variables
23:         index.Mem ← 1
24:         index.Sto ← 1
25:         cpuStart ← 1                                   ▷ Initialise start slot numbers
26:         memStart ← 1
27:         stoStart ← 1
28:         ITres ← [ ]                                    ▷ Initialise IT resource 'tracker'
29:         for each slot s from cpuStart in C.CpuResources do
30:             if found.Cpu < R.Cpu then                  ▷ Find cpu slots
31:                 units ← C.CpuResources(s)
32:                 found.Cpu ← found.Cpu + units
33:                 ITres[1, index.Cpu] ← {s, units}
34:                 index.Cpu ← index.Cpu + 1
35:             else
36:                 break
37:             end if
38:         end for
39:         for each slot s from memStart in C.MemResources do
40:             if found.Mem < R.Mem then                  ▷ Find memory slots
41:                 units ← C.MemResources(s)
42:                 found.Mem ← found.Mem + units
43:                 ITres[2, index.Mem] ← {s, units}
44:                 index.Mem ← index.Mem + 1
45:             else
46:                 break
47:             end if
48:         end for
49:         for each slot s from stoStart in C.StoResources do
50:             if found.Sto < R.Sto then                  ▷ Find storage slots
51:                 units ← C.StoResources(s)
52:                 found.Sto ← found.Sto + units
53:                 ITres[3, index.Sto] ← {s, units}
54:                 index.Sto ← index.Sto + 1
55:             end if
56:         end for
57:         if R.Cpu and R.Mem and R.Sto found then        ▷ All resources found
58:             ITallocation ← Success
59:             ITfailureCause ← None
60:             break
61:         else
62:             ITallocation ← Failure
63:             Evaluate failure cause
64:             Update ITfailureCause and ITcheckBreak
```

```
65:             end if
66:             if ITallocation = Success then
67:                 [NETallocation, NETres] ← netAllocation(G, C, R, ITres)
68:                 if NETallocation = Success then
69:                     break
70:                 else
71:                     Remove failure nodes in C.CpuResources
72:                     Remove failure nodes in C.MemResources
73:                     Remove failure nodes in C.StoResources
74:                 end if
75:             else
76:                 if ITcheckBreak = 1 then
77:                     break                ▷ Failed due to unavailability of IT resources
78:                 end if
79:             end if
80:             cpuStart ← cpuStart + loopIncrement          ▷ Increment all slot indices
81:             memStart ← memStart + loopIncrement
82:             stoStart ← stoStart + loopIncrement
83:             slot ← slot + loopIncrement                  ▷ Jump to next rack
84:         end while
85:     end if
86:     if ITallocation = Success and NETallocation = Success then
87:         Update C.ITres                          ▷ Remove allocated IT resources
88:         Update G.bMap                       ▷ Remove allocated network resources
89:     end if
90:     return ITallocation, ITres, NETallocation, NETres
91: end procedure
```

4.5 Network-Unaware Locality Based Resource Allocation

The network-unaware locality based algorithm uses the breadth-first search (BFS) algorithm to allocate IT resources. The advantage of using BFS is that it can find IT resource nodes that are neighbouring each other (making it locality aware); this can be particularly useful with requests that have low latency constraints. The algorithm starts scanning for IT resources on a node of a specific resource type that has the highest contention ratio (CR) and looks for other IT resource nodes neighbouring this node. The contention ratio for each resource types is simply the ratio between the units required (for a request) and the total units available in the data centre. Using contention ratios hugely improves performance as it first scans and allocates resources for which the least number of units are available, making it more likely for a request to be successfully allocated its required resources. In comparison to the first fit and best fit algorithm, a huge improvement in performance was observed, resulting in higher utilisation and lower blocking probability. The pseudocode for the network-unaware locality based algorithm is provided in Algorithm 9. Once again, the network resource allocation algorithm used is the one provided in Algorithm 7.

Algorithm 9: Network-unaware locality based resource allocation algorithm

Require: G: Data centre graphs, C: Data centre config, R: Request

1: **procedure** NULOCALITYBASED(G, C, R)
2: $available.Cpu \leftarrow$ find slots $\geqslant 1$ in $C.CpuLocations$
3: $available.Mem \leftarrow$ find slots $\geqslant 1$ in $C.MemLocations$
4: $available.Sto \leftarrow$ find slots $\geqslant 1$ in $C.StoLocations$
5: $cr.Cpu \leftarrow \frac{R.Cpu}{available.Cpu}$ ▷ Cpu contention ratio
6: $cr.Mem \leftarrow \frac{R.Cpu}{available.Mem}$ ▷ Memory contention ratio
7: $cr.Sto \leftarrow \frac{R.Cpu}{available.Sto}$ ▷ Storage contention ratio
8: $cr.Max \leftarrow max(cr.Cpu, cr.Mem, cr.Sto)$ ▷ Find maximum contention ratio
9: $loopIncrement \leftarrow C.nSlots \times C.nBlades$ ▷ Size of a rack
10: **for each** c in cr **do** ▷ To try multiple contention ratios
11: $s \leftarrow cr.Max$
12: $cr.Max \leftarrow$ Update to new maximum contention ratio
13: **switch** s **do**
14: **case** Cpu
15: **if** $available.Cpu < R.Cpu$ **then** ▷ Inadequate cpu resources
16: $ITallocation \leftarrow Failure$
17: $ITfailureCause \leftarrow Cpu$
18: **else**
19: **while** $slot \leqslant C.CpuResources$ **do**
20: $[ITallocation, ITres] \leftarrow$ BFS$(G, slot, R)$
21: **if** $ITallocation = Success$ **then**
22: $[NETallocation, NETres] \leftarrow$ NETALLOCATION$(G, C, R, ITres)$
23: **if** $NETallocation = Success$ **then**
24: **break**
25: **else**
26: Remove failure nodes in $G.ITres$
27: **end if**
28: **else**
29: Evaluate failure cause
30: Update $ITfailureCause$
31: **break**
32: **end if**
33: $slot \leftarrow slot + loopIncrement$
34: **end while**
35: **end if**
36: **if** $ITallocation = Success$ and $NETallocation = Success$ **then**
37: **break**
38: **end if**
39: **case** Mem
40: **if** $available.Mem < R.Mem$ **then** ▷ Inadequate memory resources
41: $ITallocation \leftarrow Failure$
42: $ITfailureCause \leftarrow Mem$
43: **else**
44: **while** $slot \leqslant C.MemResources$ **do**
45: $[ITallocation, ITres] \leftarrow$ BFS$(G, slot, R)$
46: **if** $ITallocation = Success$ **then**

```
47:                         [NETallocation, NETres] ← netAllocation(G, C, R, ITres)
48:                         if NETallocation = Success then
49:                             break
50:                         else
51:                             Remove failure nodes in G.ITres
52:                         end if
53:                     else
54:                         Evaluate failure cause
55:                         Update ITfailureCause
56:                         break
57:                     end if
58:                     slot ← slot + loopIncrement
59:                 end while
60:             end if
61:             if ITallocation = Success and NETallocation = Success then
62:                 break
63:             end if
64:         case Sto
65:             if available.Sto < R.Sto then          ▷ Inadequate storage resources
66:                 ITallocation ← Failure
67:                 ITfailureCause ← Sto
68:             else
69:                 while slot ⩽ C.StoResources do
70:                     [ITallocation, ITres] ← BFS(G, slot, R)
71:                     if ITallocation = Success then
72:                         [NETallocation, NETres] ← netAllocation(G, C, R, ITres)
73:                         if NETallocation = Success then
74:                             break
75:                         else
76:                             Remove failure nodes in G.ITres
77:                         end if
78:                     else
79:                         Evaluate failure cause
80:                         Update ITfailureCause
81:                         break
82:                     end if
83:                     slot ← slot + loopIncrement
84:                 end while
85:             end if
86:             if ITallocation = Success and NETallocation = Success then
87:                 break
88:             end if
89:         if ITallocation = Failure then             ▷ Inadequate IT resources
90:             break
91:         end if
92:     end switch
93: end for
94: if ITallocation = Success and NETallocation = Success then
95:     Update C.ITres                                 ▷ Remove allocated IT resources
```

96: Update $G.bMap$ ▷ Remove allocated network resources
97: **end if**
98: **return** $ITallocation$, $ITres$, $NETallocation$, $NETres$
99: **end procedure**

4.6 Network-Aware Locality Based Resource Allocation

The network-aware locality based algorithm is similar to the network-aware locality based algorithm but it uses the modified breadth-first search (mBFS) algorithm. The advantage of using the mBFS algorithm is that it finds and allocates IT resource nodes that are not only neighbouring each other, but also considers the network resources connecting these nodes. Considering the network resources when allocating IT resources nodes improves performance as it significantly reduces the probability of failure when allocating network resources. The pseudocode for mBFS is provided in Algorithm 10. The rest of the algorithm is the same as the network-unaware locality based algorithm, except on lines 20, 45, and 70 (in Algorithm 9), instead of invoking the standard BFS method, the modified-BFS (mBFS) method is invoked. The results for this algorithm are discussed in the Results section.

Algorithm 10: Modified breadth-first search (mBFS) algorithm

Require: G: Data centre graphs, s: Start vertex, R: Request
1: **procedure** mBFS(G, s, R)
2: $bMap \leftarrow G.bMap$ ▷ Copy original bandwidth map
3: $dMap \leftarrow G.dMap$ ▷ Copy original distance map
4: Remove all edges in $bMap$ and $dMap$ where $e_b < min(R.b_{cm}, R.b_{ms})$
5: **for each** vertex v in G **do** ▷ Initialise each vertex
6: $v.distance \leftarrow \infty$
7: $v.parent \leftarrow null$
8: **end for**
9: $Q \leftarrow [\,]$ ▷ Create an empty queue
10: $s.distance \leftarrow 0$
11: $Q.enqueue(s)$ ▷ Enqueue start vertex
12: $ITres \leftarrow$ Find IT resources on start vertex s
13: **while** $Q \neq [\,]$ **do** ▷ Run until queue is empty
14: $current \leftarrow Q.dequeue()$
15: $neighbours \leftarrow$ All vertices adjacent to $current$
16: ▷ High bandwidth links have a higher priority (Network aware)
17: $neighbours' \leftarrow$ Sort $neighbours$ in descending order
18: **for each** vertex v in $neighbours'$ **do**
19: **if** $v.distance = \infty$ **then**
20: $v.distance \leftarrow current.distance + 1$
21: $v.parent \leftarrow current$
22: $Q.enqueue(v)$ ▷ Enqueue v^{th} neighbour
23: $ITres \leftarrow$ Find IT resources on vertex v
24: **if** $R.Cpu$ and $R.Mem$ and $R.Sto$ found **then** ▷ All resources found

```
25:                 breakWhile ← True
26:                 break
27:             end if
28:         end if
29:     end for
30:     if breakWhile = True then
31:         ITallocation ← Success
32:         ITfailureCause ← None
33:         break
34:     else
35:         ITallocation ← Failure
36:         ITfailureCause ← Evaluate failure cause
37:         break
38:     end if
39: end while
40: return ITres, ITallocation
41: end procedure
```

5 Results

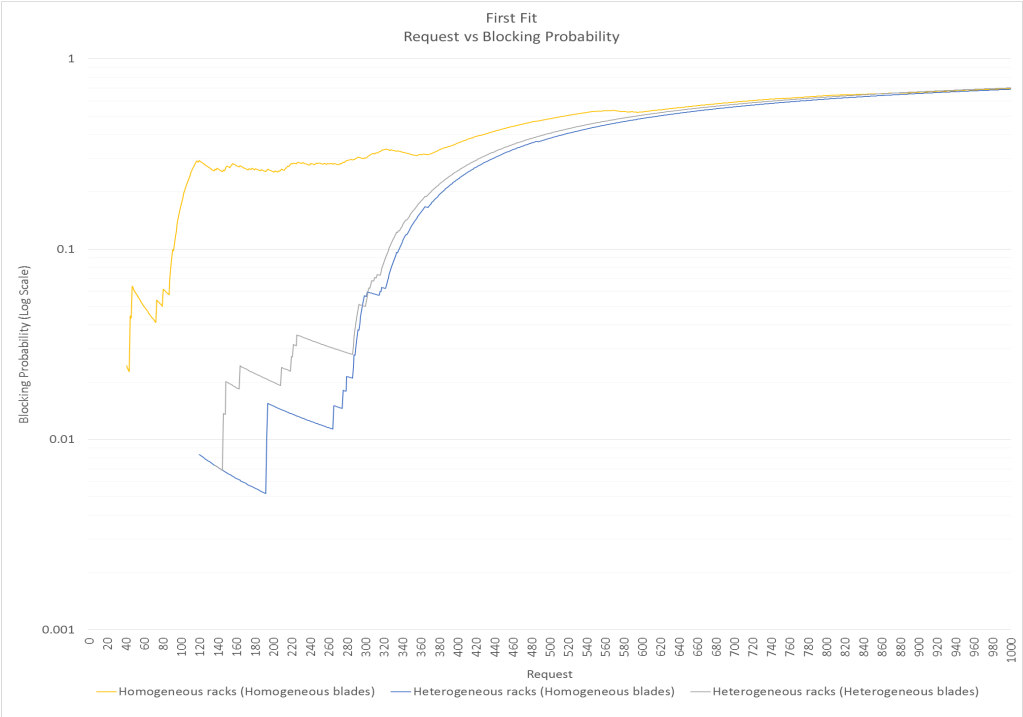
The results discussed are sectioned based on different performance metrics. These include blocking probability, IT & network utilisation, allocation time, latency, and an overall performance comparison. It is important to note that to make a fair comparison between the performance of all algorithms across all architectures a predefined request database, i.e. a set of dynamic requests, was used during the simulations. Both static and dynamic requests were seen to have a very similar performance.

5.1 Blocking Probability

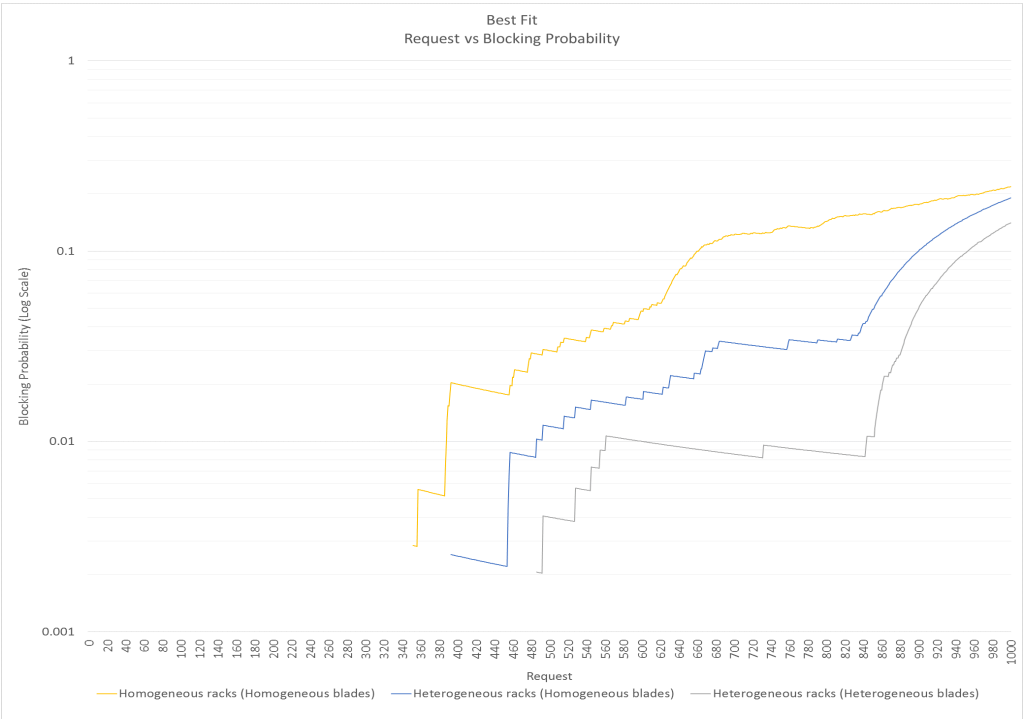
Blocking probability is the ratio between the number of blocked requests and the total number of requests generated up to a certain point in time. It is considered to be a standard performance metric for allocation algorithms, where a low blocking probability implies that an algorithm is efficient. As discussed previously, there is a significant drop in blocking probability when moving from the first fit algorithm up to the network-aware locality based algorithm and this is illustrated in the blocking probability graphs in Figure 19. For every algorithm, the homogeneous rack with homogeneous blade architecture has the highest blocking probability in comparison to all other architectures simulated. This is primarily due to the higher requirement of network resources, i.e. bandwidth, for connecting the IT resources nodes – for every connection between nodes, multiple extra links are required in comparison to the heterogeneous rack with homogeneous blade and heterogeneous rack with heterogeneous blade architectures. This additional bandwidth requirement/allocation leads to a higher network resource utilisation (for every request), which in turn results in a higher blocking probability as subsequent requests have a lower amount of network resources available. Most requests that are blocked due to unavailability of network resources are caused by the lack of bandwidth on links connecting switches between different levels in the network hierarchy. Links connecting different switches, both within and between hierarchies, such as ToR-ToR and ToR-ToB, etc. tend to saturate far more quickly as compared to other links. In both the network-unaware locality based and network-aware locality based algorithms, heterogeneous rack with homogeneous blade, and heterogeneous rack with heterogeneous blade architectures have identical blocking probabilities, but if the CPU-memory latency constraint by requests were to get more strict, the latter is likely to have a better performance, i.e. a lower blocking probability.

In terms of blocking probability, the network-aware locality based algorithm has the best performance in comparison to all other algorithms, with the first request being blocked at approximately 810 for the homogeneous rack with homogeneous blade architecture, and approximately 925 for both the heterogeneous rack with homogeneous blade and heterogeneous rack with heterogeneous blade architectures.

Although disaggregated architectures allow for dynamic bandwidth allocation, the primary bottleneck in such architectures can be bandwidth and latency, both because of the resources being placed far apart (in different racks). The cause for requests being

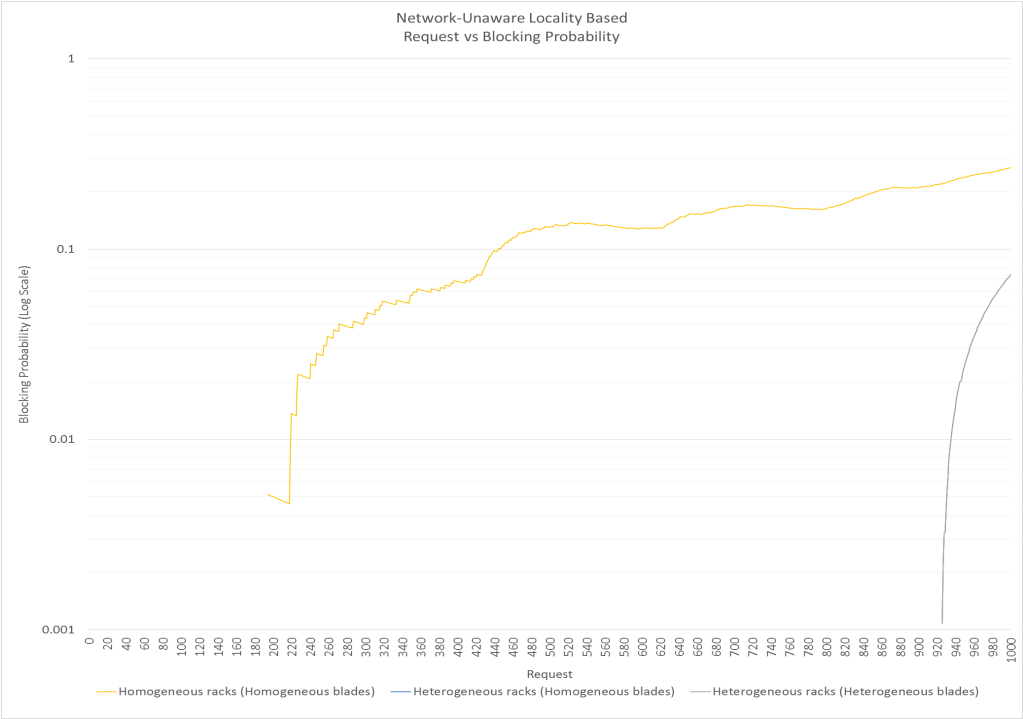


(a) *First fit.*

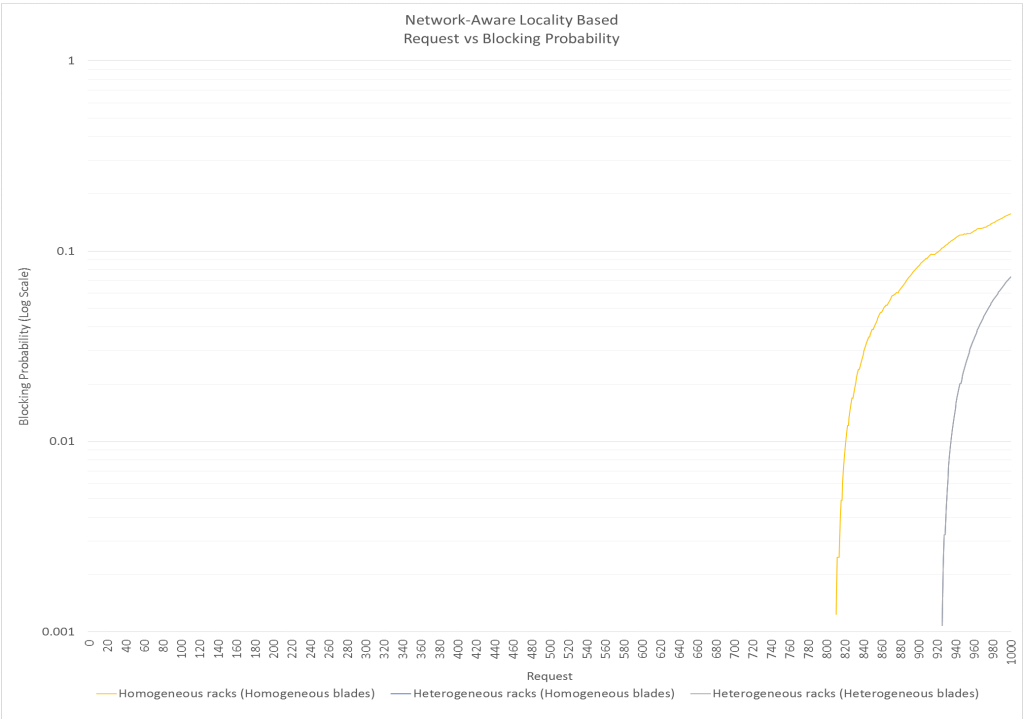


(b) *Best fit.*

blocked for all algorithms across all architectures is illustrated in Figure 20. It can be seen that majority of the requests are blocked due to unavailability of network resources, i.e. bandwidth. There are a few requests that are blocked due to latency – this signifies that the algorithms were unable to find IT resource nodes and/or connections between these



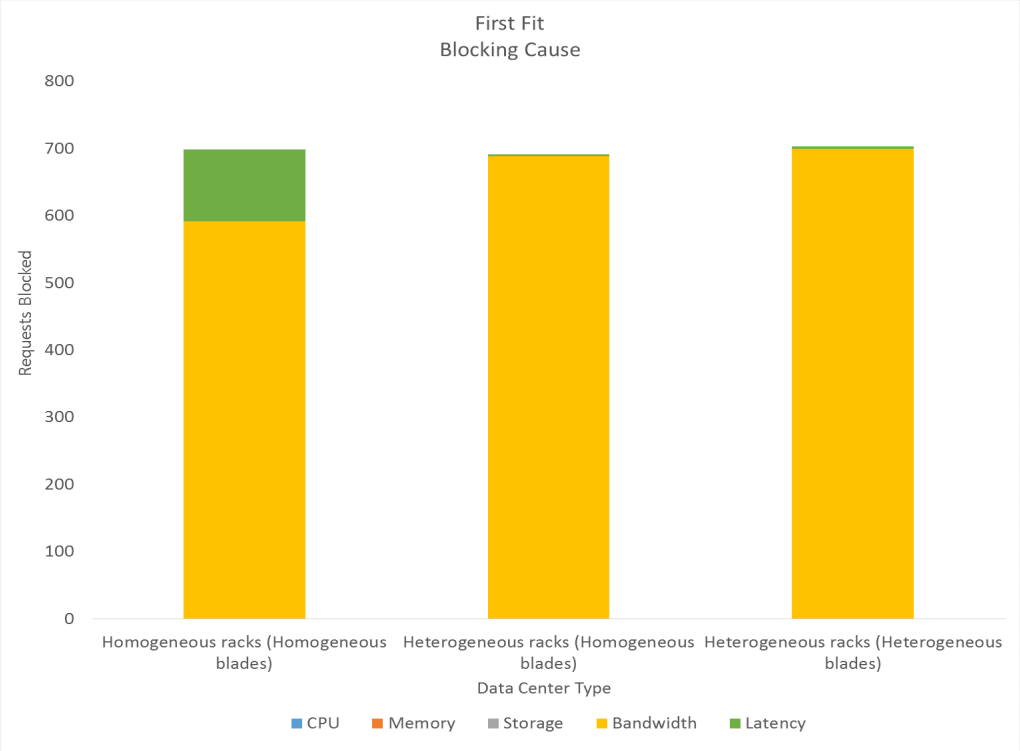
(c) Network unaware locality based.



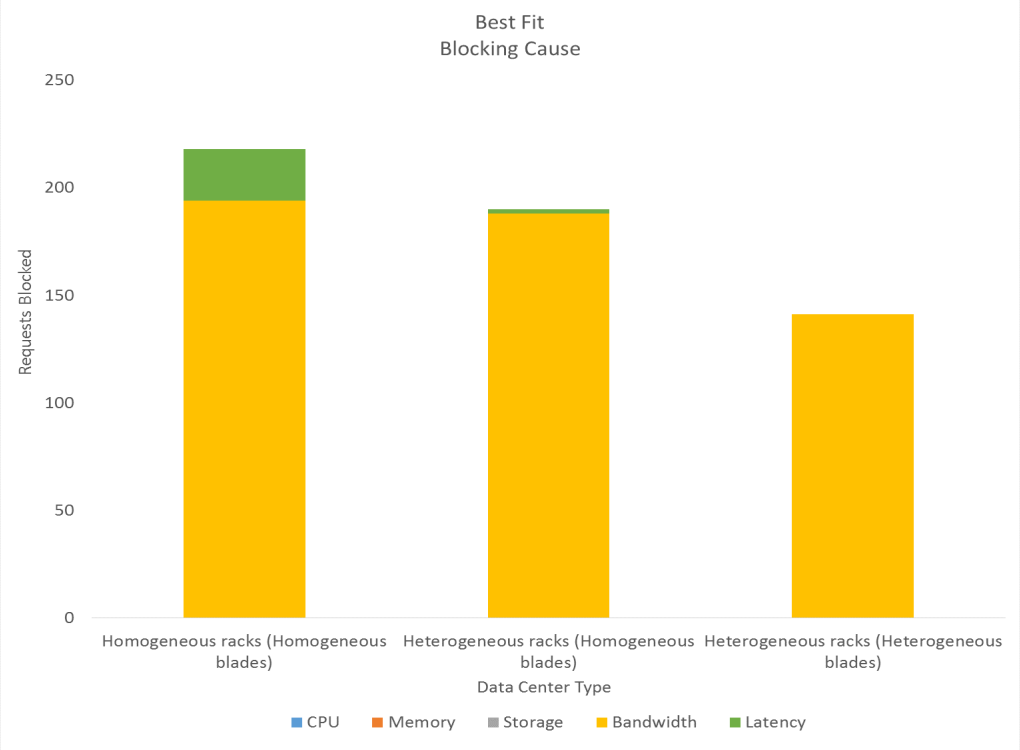
(d) Network aware locality based.

Figure 19: *Comparison between blocking probabilities obtained from different resource allocation algorithms.*

nodes that satisfied the latency constraints. This is more frequent in the homogeneous rack with homogeneous blade architecture due to the different types of resources being

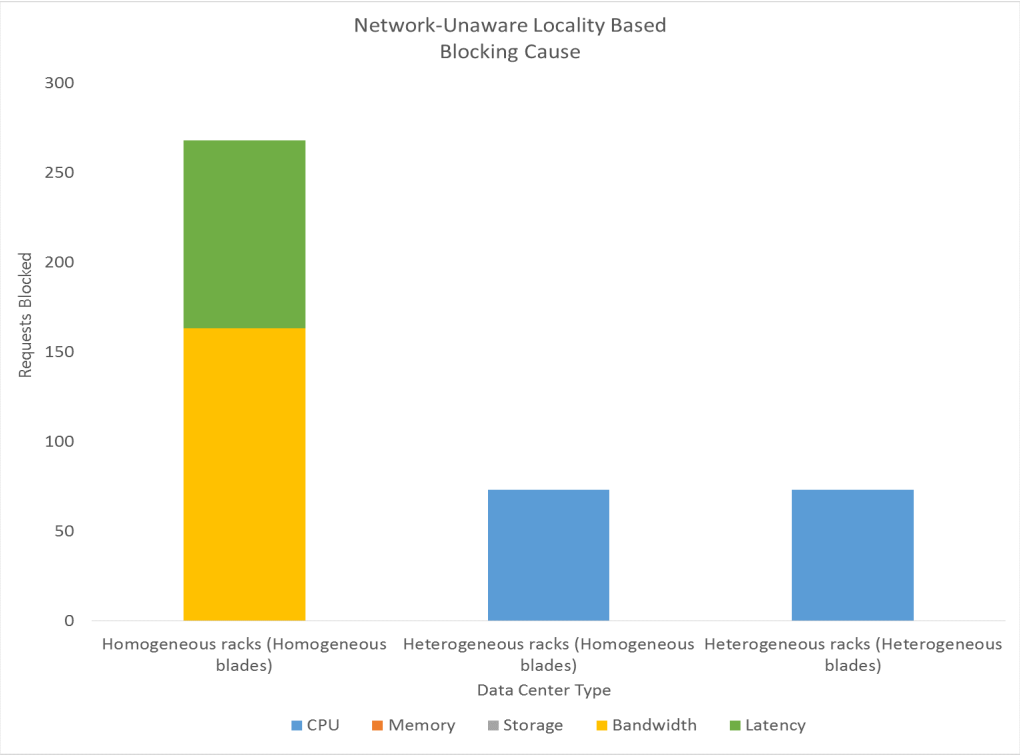


(a) *First fit.*

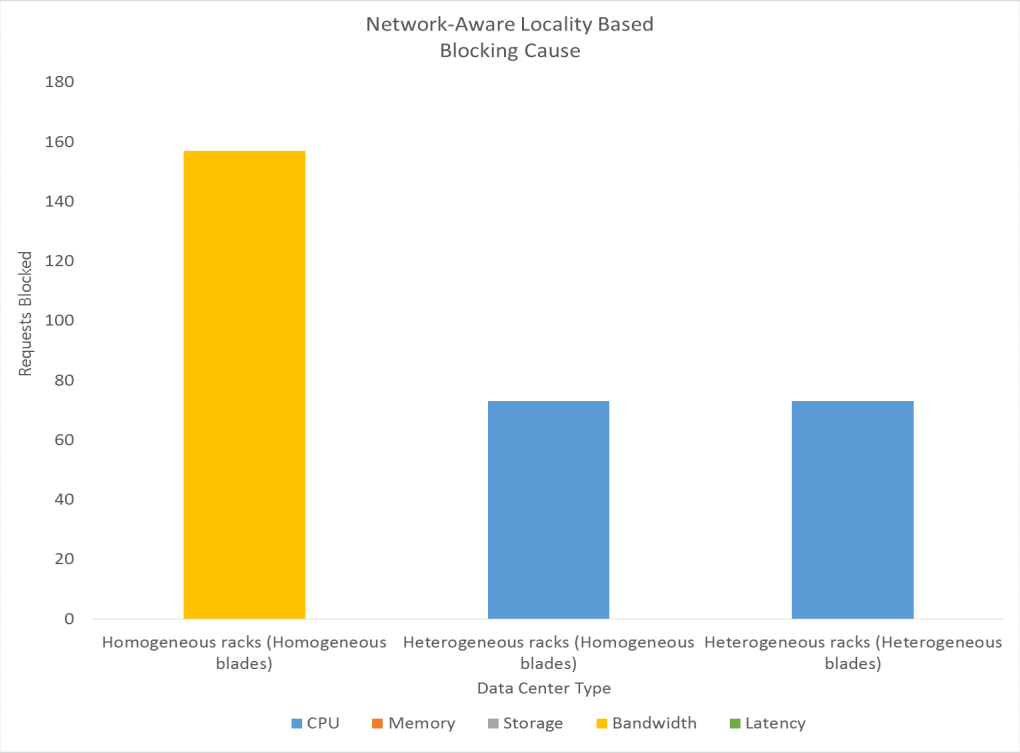


(b) *Best fit.*

placed far apart (in different racks). It is obvious that failures due to unavailability of network resources (bandwidth and/or latency) gets more frequent as the network utilisation



(c) Network unaware locality based.



(d) Network aware locality based.

Figure 20: *Comparison between blocking causes obtained from different resource allocation algorithms.*

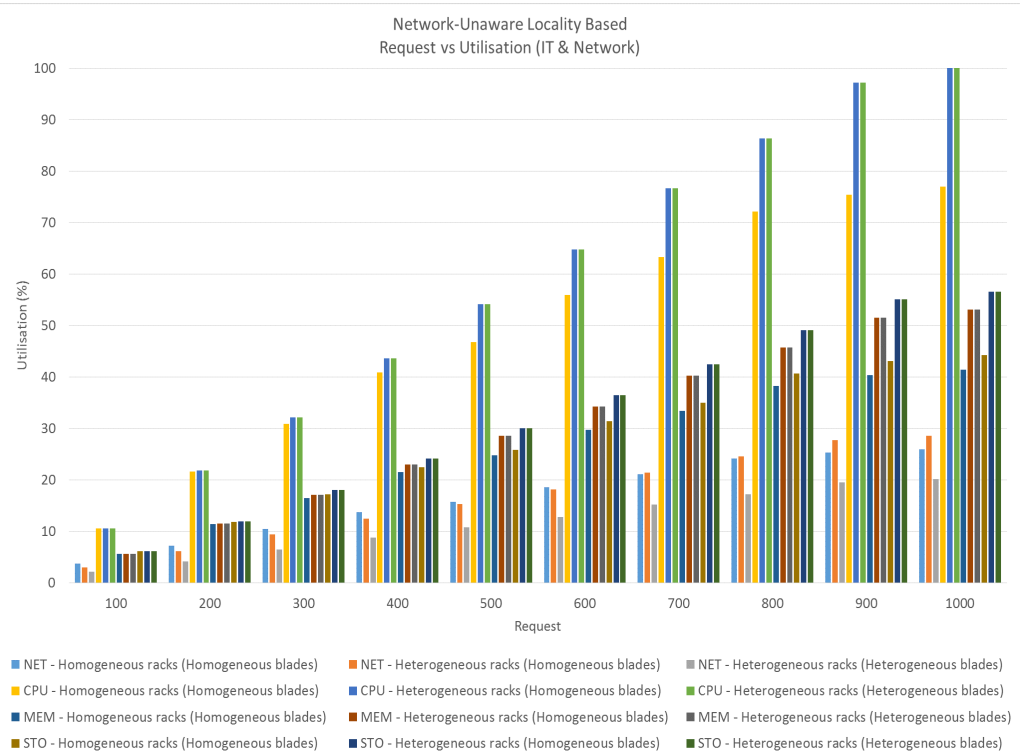
increases and the network starts to saturate. When simulating both the network-unaware locality based and network-aware locality based algorithms for the heterogeneous rack with homogeneous blade, and heterogeneous rack with heterogeneous blade architectures, all requests that were blocked were due to the unavailability of CPUs. This signifies that both the locality based algorithms are more efficient in comparison to the first fit and best fit algorithms; these algorithms were able to completely utilise the IT resources (CPUs in these results) for these architectures with no failures/blocking being caused due to the unavailability of network resources. An important point to note is that there are infinitely many possible combinations, for allocating both IT resource nodes and network resources, but the algorithms developed try only a very small subset of these combinations as trying all combinations is computationally infeasible. An algorithm can be said to be relatively more efficient if it can allocate a set of requests using less tries, i.e. attempts with different combinations, in comparison to another algorithm.

5.2 IT & Network Utilisation

(a) *First fit.*

As the primary goal of the algorithms developed was to maximise IT resource utilisation whilst minimising network utilisation, graphs illustrating this have been provided in Figures 21 and 22. Figure 21 shows the utilisation of both IT (CPU, memory, and storage) and network resources at intervals of every hundred requests for all algorithms, whereas Figure 22 shows the relation between network utilisation and IT resource utilisation for all algorithms. Once again, it can be seen that the network-aware locality based algorithm has

(b) Best fit.



(c) Network unaware locality based.

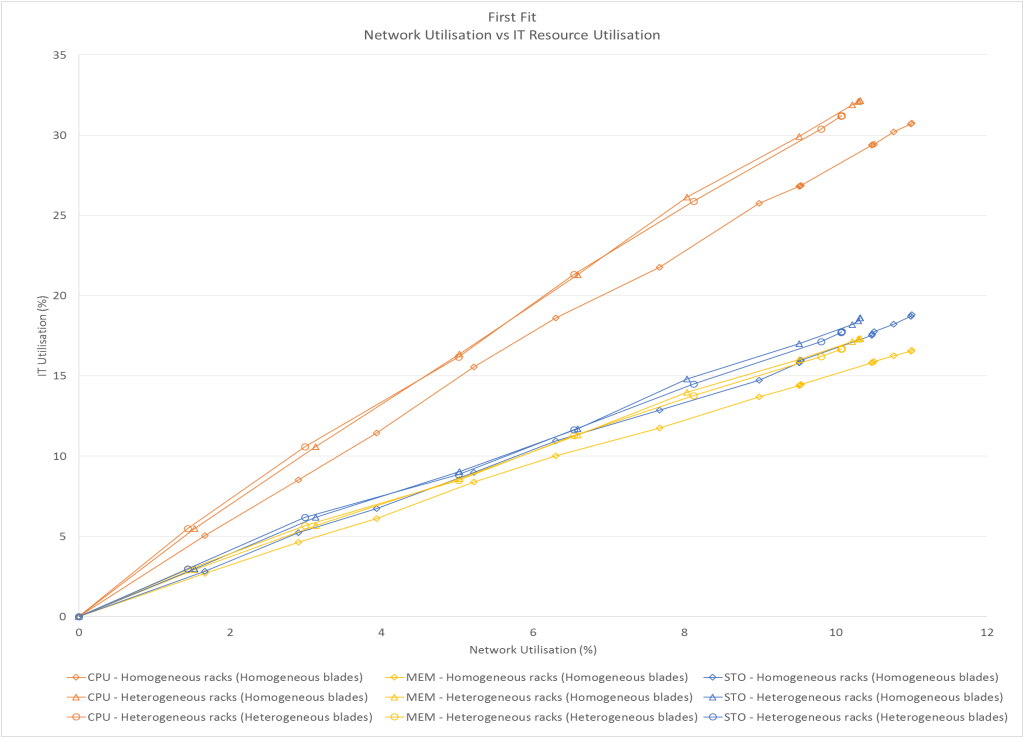
the best performance as it is able to completely utilise all CPUs whilst only utilising 16% of the network resources for the heterogeneous rack with heterogeneous blade architecture;

(d) *Network aware locality based.*

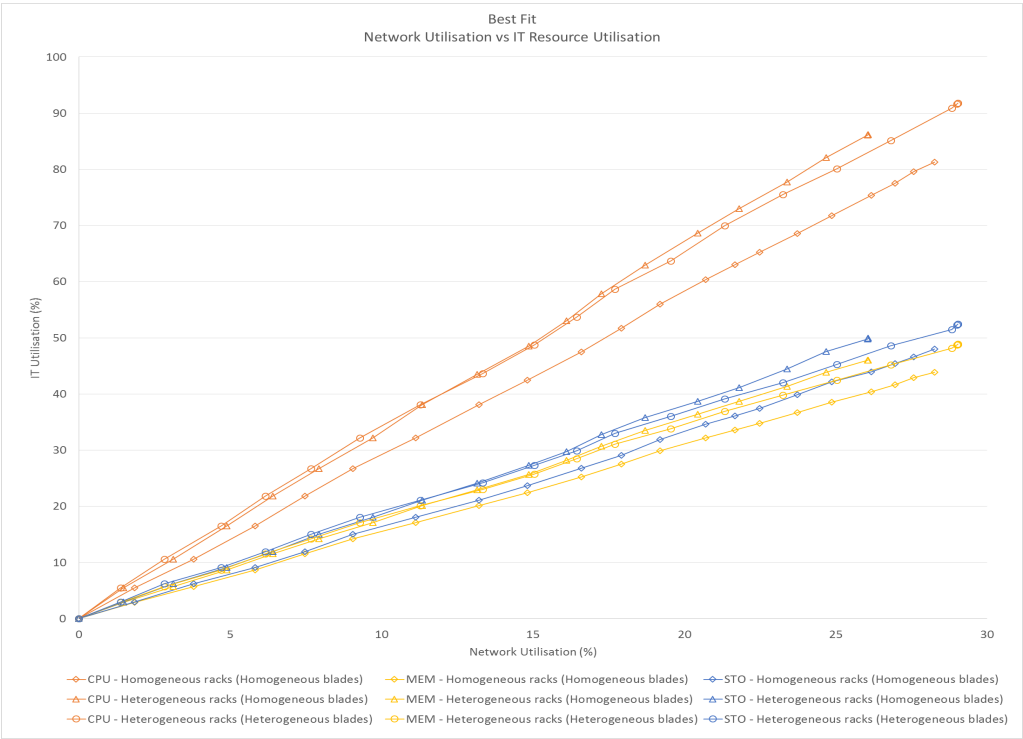
Figure 21: *Comparison between IT and network resource utilisation obtained from different resource allocation algorithms.*

a similar trend follows for the other architectures. When using the same set of requests on the network-unaware locality based algorithm, the network utilisation increases to 20%. With the first fit and best fit algorithms, the network utilisation increases to 10% and 30% whilst only utilising 32% and 92% of available CPUs respectively. An important point to note is that the network utilisation for the first fit algorithm is low in comparison to the other algorithms not because of it being more efficient in terms of network allocation but because of it having a higher blocking probability – as more requests get blocked, the network utilisation stays low. Once a type of resource is completely utilised, all subsequent requests are blocked; this is the point at which the utilisation for all resources stabilise. Achieving the maximum IT resources utilisation with a low network utilisation (and being able to successfully allocate the required resources for more requests) proves that the network-aware locality based algorithm has the best performance in terms of the goal the algorithms try to achieve. For the three different types of architectures simulated, the heterogeneous rack with heterogeneous blade was seen to have the best performance in comparison to the other architectures. An example illustrating this can be found on Figure 21(d); after a thousand requests, both the heterogeneous rack with homogeneous blade and heterogeneous rack with heterogeneous blade architectures are able to completely utilise all the CPU resources but the former has a network utilisation of 24%, whereas the latter has a network utilisation of 16%. A similar analysis can be made based

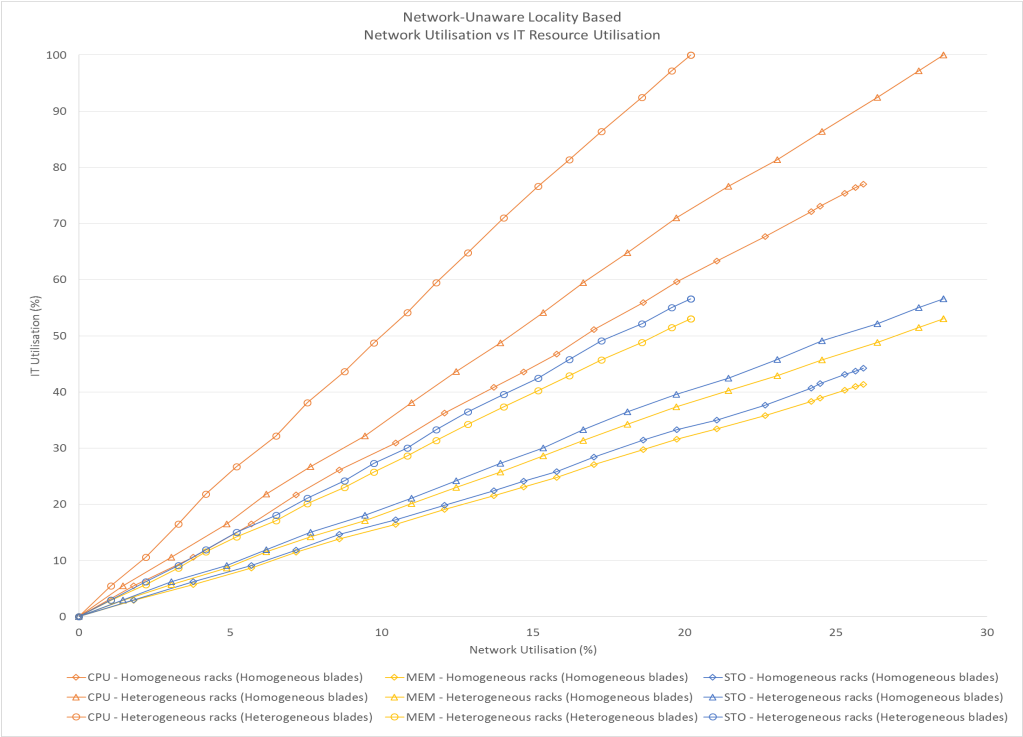
on the network utilisation versus IT resource utilisation graphs illustrated in Figure 22.



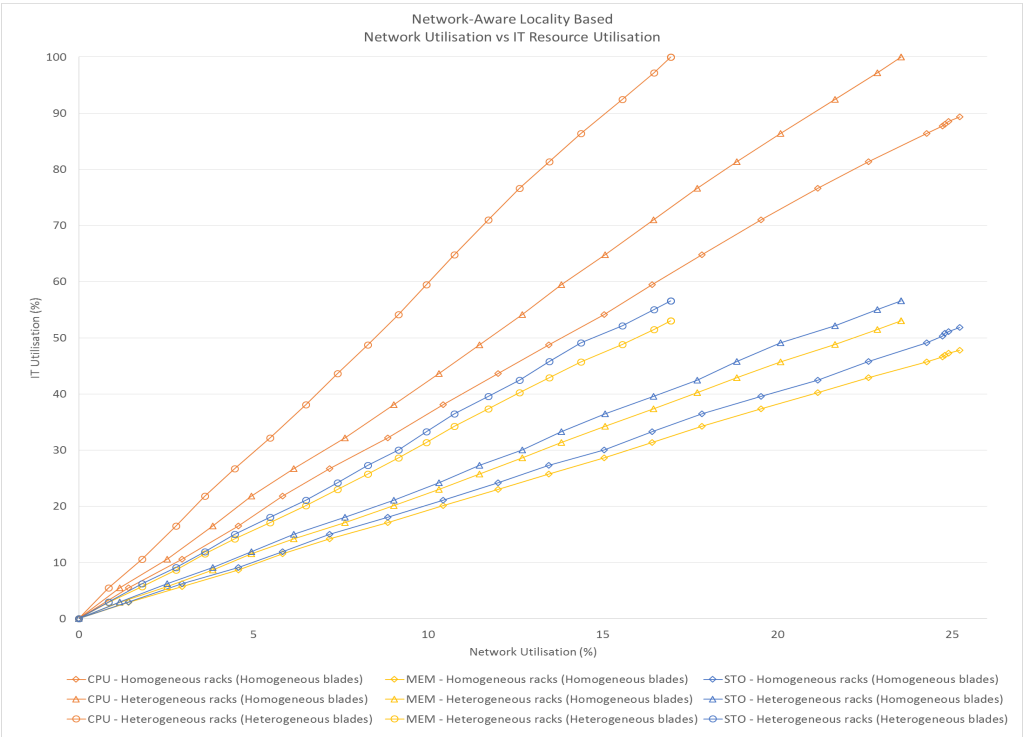
(a) *First fit.*



(b) *Best fit.*



(c) *Network unaware locality based.*

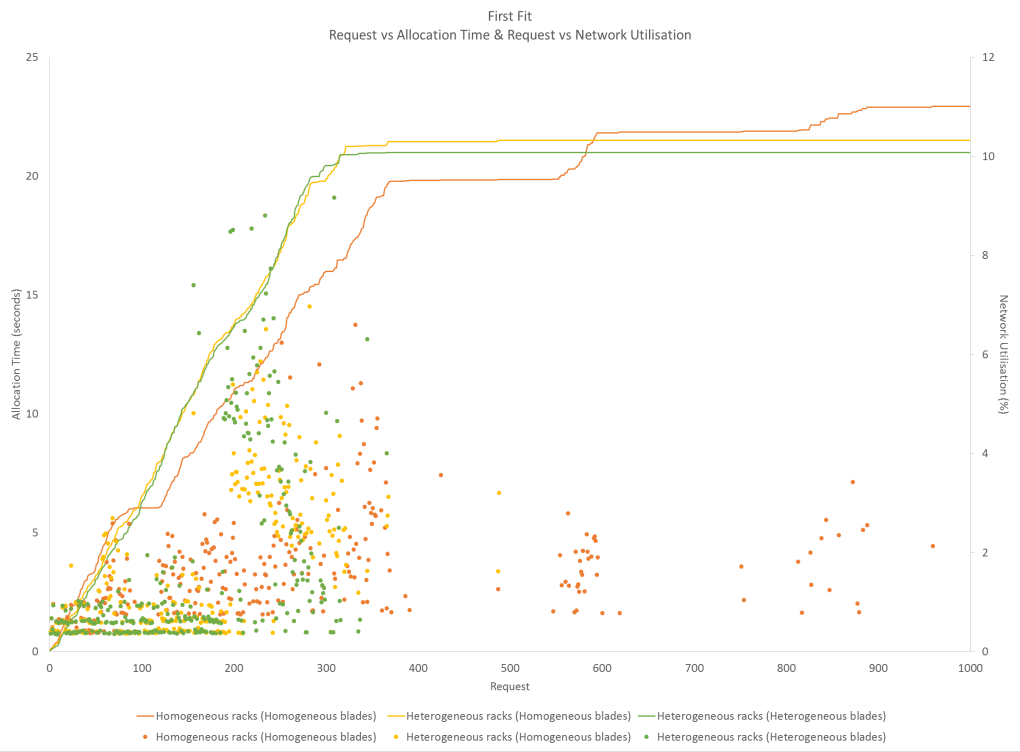


(d) *Network aware locality based.*

Figure 22: *Comparison between network vs IT resource utilisation obtained from different resource allocation algorithms.*

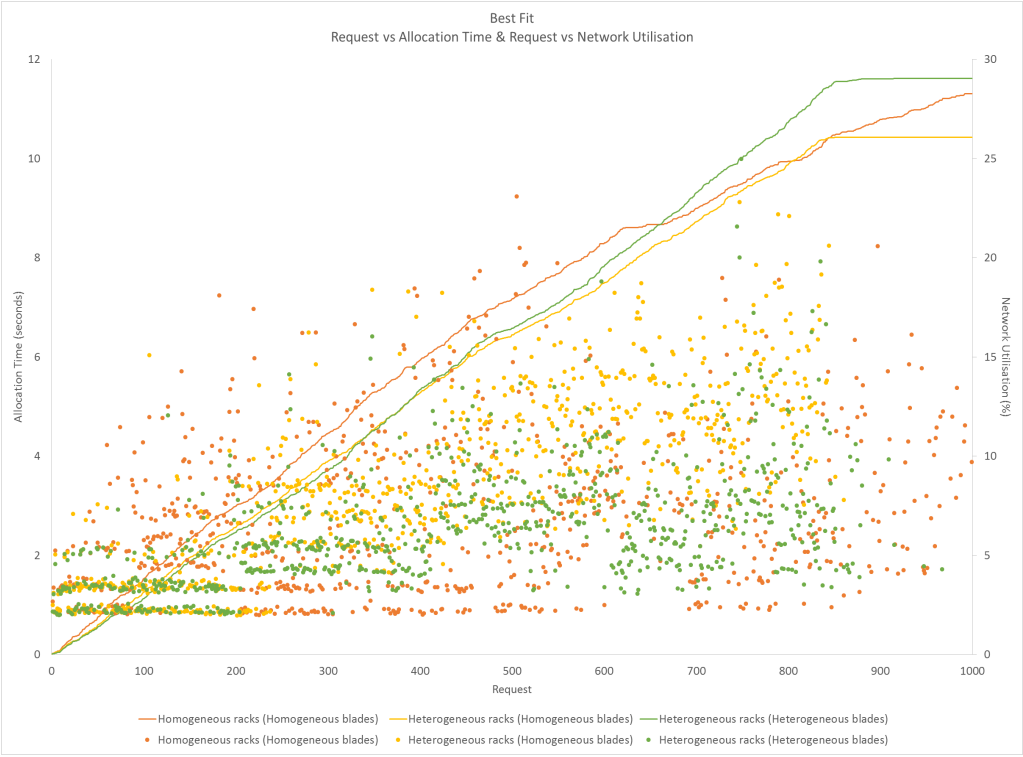
5.3 Allocation Time

The time taken of find and allocate the required resources can heavily dictate the performance of an algorithm. Using efficient searching (sorting, and scanning) algorithms and data structures; and optimising the resource allocation algorithms can significantly reduce the allocation time. Figure 23 illustrates the allocation time for all requests across all algorithms. These graphs only contain data points for requests that were successfully allocated. A common trend that can be observed for all algorithms across all architectures is that as the network utilisation increases the allocation time increases. The reason behind this is fairly obvious; due to the increase in network utilisation, certain combinations of IT resources chosen by an algorithm have a higher probability of failure during the network allocation phase, therefore, leading to an increased number of tries with different combinations, which directly affects the allocation time.

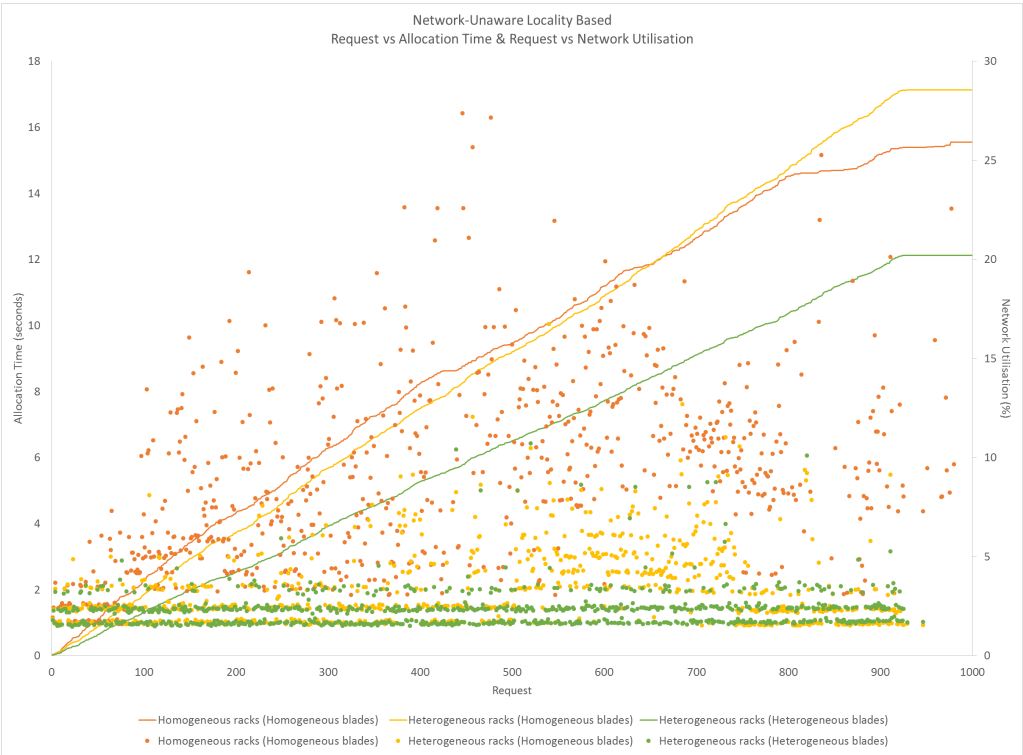


(a) *First fit.*

For the first fit algorithm, once the network utilisation reaches 5%, there is a sharp increase in the allocation time. For the other algorithms, the allocation time is not as drastically affected by the network utilisation. Once again, moving from the first fit algorithm to the network-aware locality based algorithm, there is a huge improvement in allocation time. Across all requests, the network-based locality aware algorithm has the best performance with only a few requests taking over six seconds for resources to be allocated. For majority of the requests, the allocation time on the homogeneous rack with homogeneous blade architecture drops significantly in the network-aware locality based algorithm as compared to the network-unaware locality based algorithm. The is because

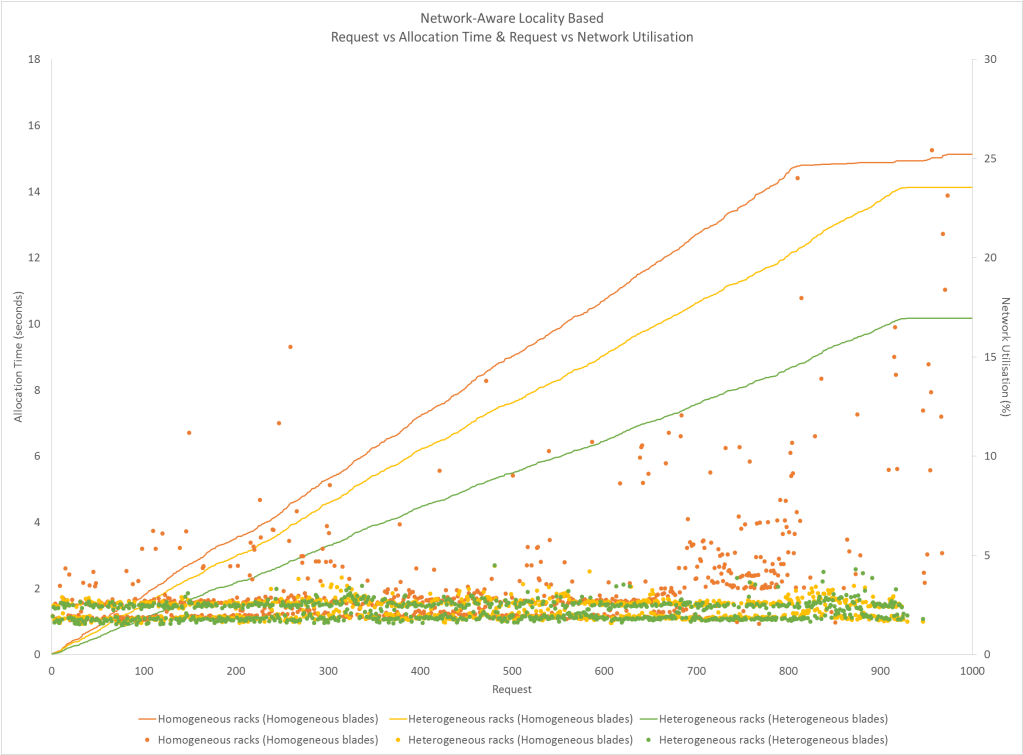


(b) *Best fit.*



(c) *Network unaware locality based.*

the network-aware locality based algorithm chooses IT resources based on the network resources available on links connecting these nodes, this drastically reduces the probability



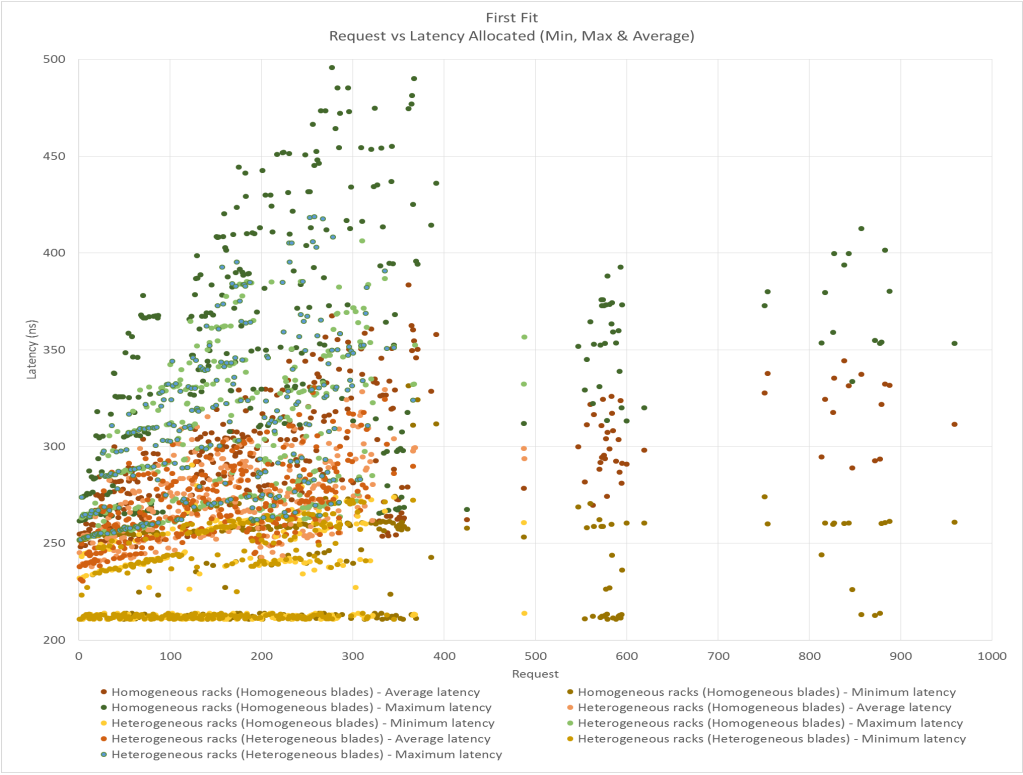
(d) *Network aware locality based.*

Figure 23: *Comparison between allocation times and network utilisation obtained from different resource allocation algorithms.*

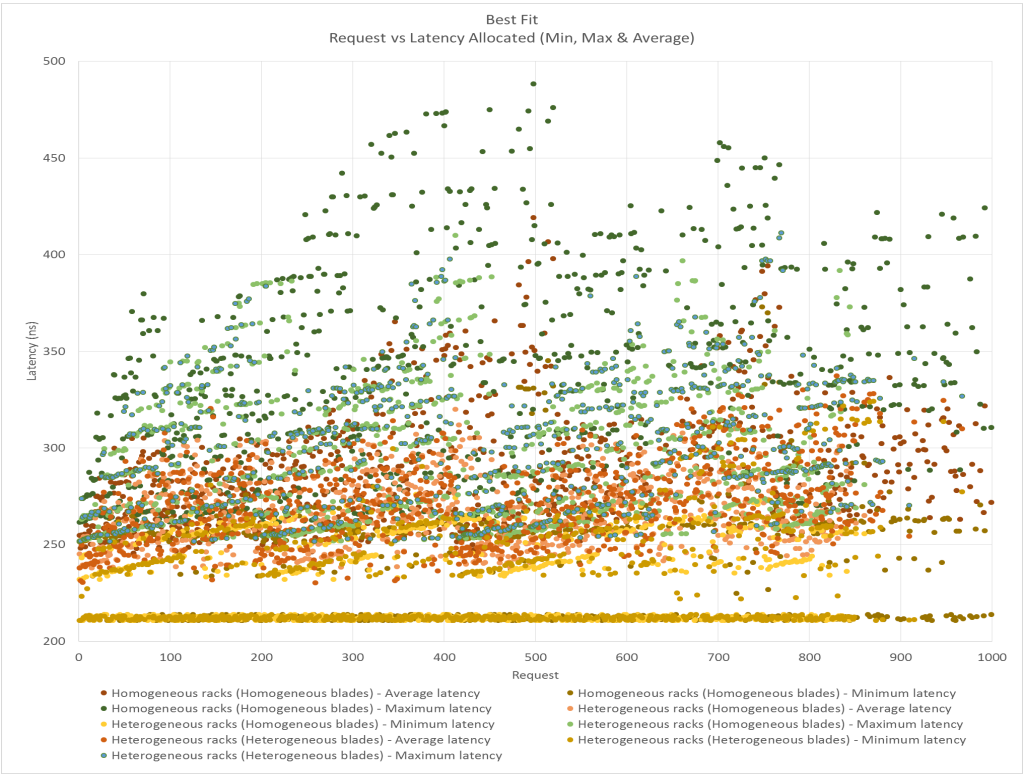
of failure during the network allocation phase. In summary, across all algorithms, the heterogeneous rack with homogeneous blade architecture has the best performance in terms of allocation time.

5.4 Latency

Latency is an important factor that needs to be considered when evaluating an algorithm, especially for disaggregated architectures as resource pooling can have a huge impact on the latencies available between (different types of) IT resources. Figure 24 illustrates scatter plots for all algorithms across all architectures that represent the latencies allocated for all requests. Different latencies such as CPU-CPU, CPU-memory, etc. have been plotted using different colours. These graphs only contain data points for requests that were successfully allocated; and data points for requested/maximum acceptable latency for each request have not been plotted not only because of all allocated latencies being way below the requested latency but also to improve the readability of graphs. It can be seen that the heterogeneous rack with heterogeneous blade architecture has the lowest latencies allocated across all algorithms; this is because CPU and memory slots are located on the same blade and connections between these slots only require a single switch. This can be beneficial for systems and applications for which low latency is critical, particularly the latency between CPU and memory. The fist fit algorithm has the highest latencies allocated with a few

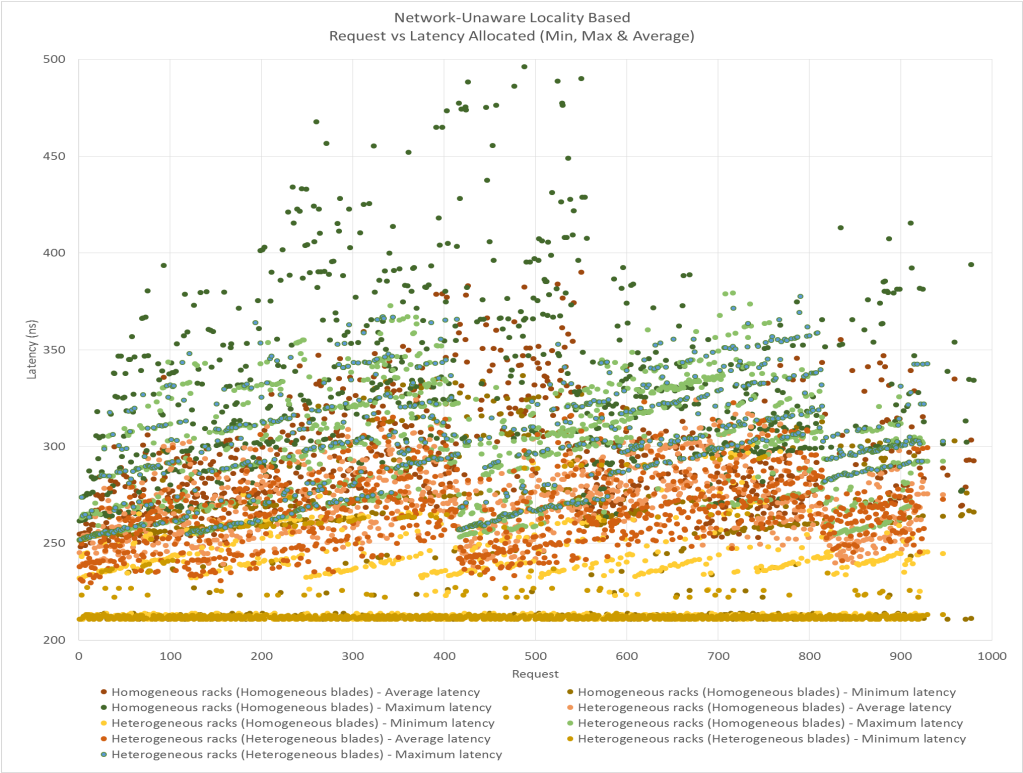


(a) *First fit.*

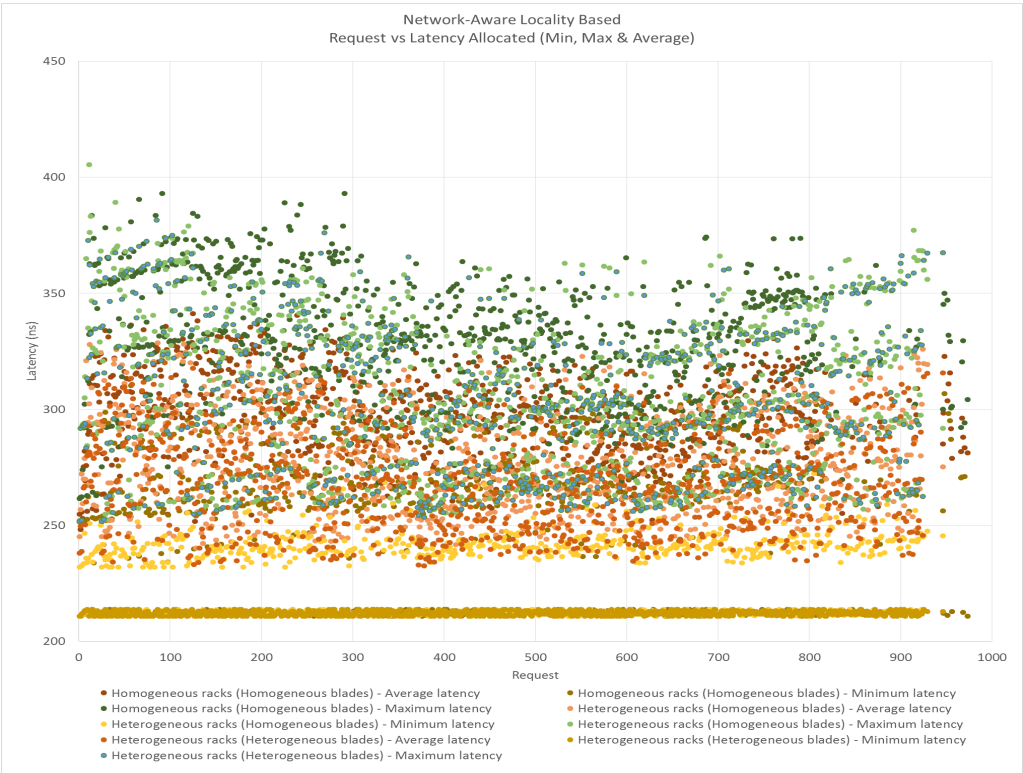


(b) *Best fit.*

reaching very close to 500 ns, whereas the best fit and network-unaware locality based

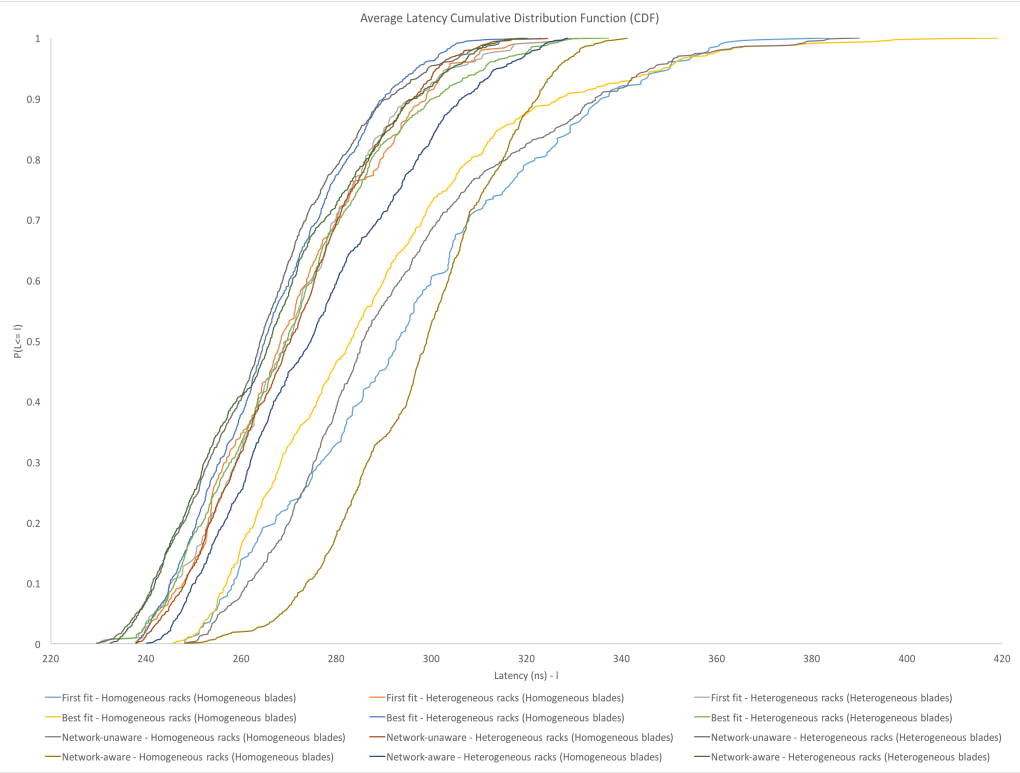


(c) Network unaware locality based.



(d) Network aware locality based.

algorithms have very similar latencies allocated. Majority of latencies allocated with the



(e) *Allocated average latency cumulative distribution function (CDF).*

Figure 24: *Comparison between latency allocations obtained from different resource allocation algorithms.*

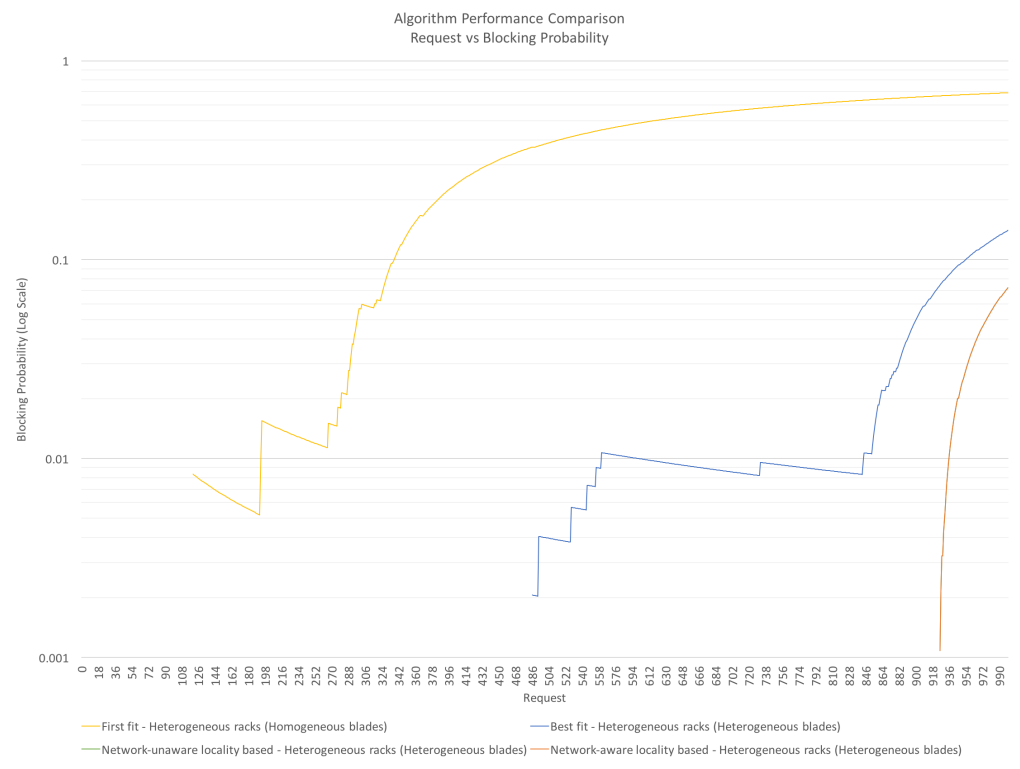
network-aware locality based algorithm are below 350 ns with only a few allocated between 350-400 ns.

Figure 24(e) illustrates a cumulative distribution function (CDF) for average latencies allocated for all algorithms across all architectures. It is important to note that distribution functions only consider average latencies of allocated requests and these have not been normalised; this is why some algorithms/architectures may appear to have a better overall distribution of average allocated latencies in comparison to other. For example, the best fit algorithm appears to have allocated better latencies in comparison to both the network-unaware and network-aware locality based algorithms for the heterogeneous rack with homogeneous blade architecture even though the former has a higher blocking probability. To make a fair comparison, both the blocking probability and latency graphs need to be analysed together.

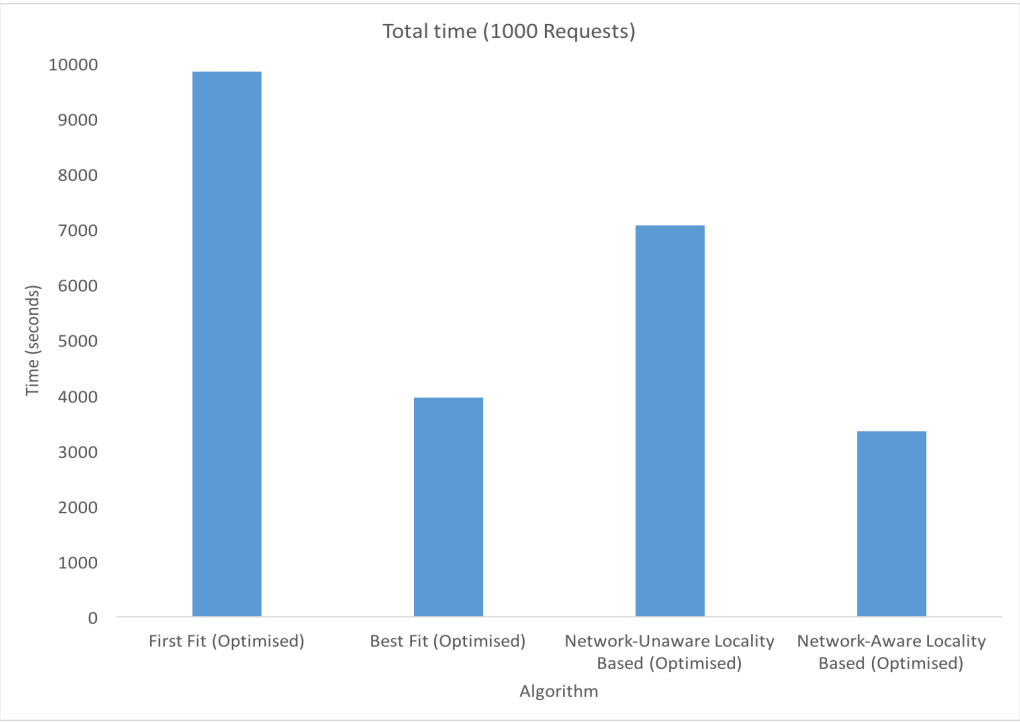
5.5 Overall Performance Comparison

Figure 25 illustrates graphs for an overall performance comparison between all algorithms, where (a) illustrates the best, i.e. lowest, blocking probability for every algorithm and (b) illustrates the total simulation time[1] for allocating resources across thousand requests. The first fit algorithm performs best on the heterogeneous rack with homogeneous blade

[1]Simulation times are platform dependent and results discussed here are based on simulations being run on MATLAB R2015b for Microsoft Windows 10.



(a) Lowest blocking probability for different algorithms.



(b) Total allocation time for 1000 requests across all algorithms.

Figure 25: *Overall performance comparison between different algorithms.*

architecture, whereas the best fit, network-unaware and network-aware locality based algorithms perform best on the heterogeneous rack with heterogeneous blade architecture. The network-aware locality based algorithm has the lowest total simulation time in comparison to all other algorithms as its probability of failure for a certain combination of resources, both IT and network, is significantly lower than the probability of failure for other algorithms. Its failure probability is low as a result of it choosing and allocating IT resource nodes based on the network resources available on links connecting these nodes. All simulation times discussed here are for the most optimised versions of the algorithms. Although the best fit and network-aware locality based algorithms have a similar simulation time, the former performs worse in terms of blocking probability, resource utilisation, and latency.

The heterogeneous rack with homogeneous blade and heterogeneous rack with heterogeneous blade architectures have an identical performance in terms of blocking probability but the former has a higher network utilisation as expected due to different types of resources being placed in separate blades. Although there is an increase in network utilisation for the homogeneous rack with homogeneous blade architecture, its performance in terms of blocking probability is only slightly worse as compared to the other architectures when using the network-aware locality based algorithm; this is primarily due to the increase in the required network resources leading to links between switches saturating far more quickly. It also tends to have higher latencies allocated in comparison to other algorithms but this is expected as different types of resources are located on different racks.

6 Conclusion

Although disaggregated architectures demand higher network resources to connect different resources in separate blades/racks, it's potential benefits in other aspects cannot be underestimated. Provided that the difference in latencies between aggregated and disaggregated architectures is marginal (with the use of super-fast interconnect technologies) and appropriate configuration of IT and network resources in disaggregated architectures such as placing both CPU and memory blades in the same rack can lead to significantly higher resource utilisation whilst yielding a similar (application) performance to that obtained in traditional data centre architectures. Disaggregation results in lower total cost of ownership and management; and dramatically improves the scope for scalability. This thesis introduced some core concepts about disaggregated data centre architectures, including several different types of disaggregated architectures, the benefits (and drawbacks) of disaggregation in general, and the potential advantages of each type of disaggregated architecture. It also discussed all the resource allocation algorithms developed, optimisation efforts and an analysis on their performance across all disaggregated architectures simulated using different performance metrics such as blocking probability, resource utilisation, etc. Based on the analysis, it was found that the network-aware locality based algorithm had the best performance. Another important finding was that when allocating IT resources, considering and evaluating the network resources available on links connecting these IT resources is crucial in increasing the probability of success, and reducing network utilisation and allocation time. In terms of the disaggregated architectures simulated, the heterogeneous rack with heterogeneous blade architecture had the best performance; the primary reason being the lower demand for network resources as both CPU and memory slots are contained in the same blade. Another potential advantage of this is that latency critical applications are likely to have better performance on this architecture in comparison to other architectures.

7 Future Work

Since disaggregated data centres is a fairly new concept, there is an incredible amount of scope for future work. The algorithms developed could be modified and improved to consider a more realistic environment that includes both inter-arrival rates and holding times. Since realistic requirements of virtual machines have a finite period for which resources are allocated, modelling these features could show the dynamic performance of disaggregated architectures and the algorithms developed. The dynamic behaviour would impact resource utilisation and based of an analysis, algorithms could be optimised and configurations of IT resources could be modified to improve overall performance. A basic implementation of inter-arrival rate has already been modelled in the simulator, but to be able to completely analyse the dynamic behaviour of the architecture/algorithm holding time would need to be modelled. General code optimisation such as the use of appropriate data structures, reduced memory access, etc. could potentially improve the performance of the algorithms. Dynamic allocation is another area that could be considered to improve the performance, where allocated resources for certain requests could be moved/transferred to accommodate new requests if required – this could lead to lower blocking probabilities and higher IT resource utilisation. The simulator could be modified to consider dropped requests after a certain amount of time to check if they can be allocated. Modelling caches, i.e. cache memories, is another area that could be looked into. Designing a latency aware algorithm could benefit latency-critical applications/requests and could potentially increase the overall resource utilisation. Simulations could be run for different network topologies, switch latencies/delays, switch configurations, CPU-memory distribution in the heterogeneous rack with heterogeneous blade architecture, etc. to evaluate the change in the performance of the algorithms/architectures.

References

[1] Intel Corporation. *Intel® Rack Scale Architecture: Faster Service Delivery and Lower TCO*. URL: http://www.intel.co.uk/content/www/uk/en/architecture-and-technology/intel-rack-scale-architecture.html (Accessed: April 19, 2016) (see pp. 1, 3, 7).

[2] Tencent and Intel Corporation. *Tencent Explores Datacenter Resource-Pooling Using Intel® Rack Scale Architecture (Intel® RSA)*. URL: http://www.intel.co.uk/content/dam/www/public/us/en/documents/white-papers/rsa-tencent-paper.pdf (Accessed: April 1, 2016) (see pp. 2, 5–8, 10, 11).

[3] Michael P. Kassner. *Silicon photonics: still waiting*. July 2015. URL: http://www.datacenterdynamics.com/it-networks/silicon-photonics-still-waiting/94303.fullarticle (Accessed: April 15, 2016) (see pp. 2, 3).

[4] Intel Corporation. *Moving data with silicon and light*. URL: http://www.intel.co.uk/content/www/uk/en/research/intel-labs-silicon-photonics-research.html (Accessed: April 14, 2016) (see pp. 1, 3, 12).

[5] Wim Bogaerts, Martin Fiers, and Pieter Dumon. "Design challenges in silicon photonics". In: *Selected Topics in Quantum Electronics, IEEE Journal of* 20.4 (2014), pp. 1–8 (see p. 1).

[6] Rick Ramsey. *Making hyperscale solutions available for comms service providers*. February 2016. URL: http://cloudblog.ericsson.com/blog/making-hyperscale-solutions-available-for-comms-service-providers (Accessed: April 19, 2016) (see p. 3).

[7] Sangjin Han, Norbert Egi, Aurojit Panda, Sylvia Ratnasamy, Guangyu Shi, and Scott Shenker. "Network support for resource disaggregation in next-generation datacenters". In: *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. ACM. 2013, p. 10 (see pp. 6, 7, 9).

[8] Ericsson. *Next-generation Datacenter Infrastructure*. February 2015. URL: http://www.ericsson.com/res/docs/whitepapers/next-generation-data-centers.pdf (Accessed: March 12, 2016) (see pp. 8, 10, 11).

[9] VMware. *Building a Data Center via Virtualization*. URL: http://www.vmware.com/uk/products/datacenter-virtualization (Accessed: March 18, 2016) (see p. 8).

[10] Cisco Systems. *Data Center Architecture Overview*. URL: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCInfra_1.html (Accessed: March 21, 2016) (see pp. 12, 13).

[11] Bulent Abali, Richard J Eickemeyer, Hubertus Franke, Chung-Sheng Li, and Marc A Taubenblatt. "Disaggregated and optically interconnected memory: when will it be cost effective?" In: *arXiv preprint arXiv:1503.01416* (2015) (see p. 13).

[12] Wikipedia. *Rack unit — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Rack_unit&oldid=713940169 (Accessed: March 27, 2016) (see p. 13).

[13] Wikipedia. *Bin packing problem — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Bin_packing_problem&oldid=717045280 (Accessed: April 22, 2016) (see pp. 25, 26).

[14] Silvano Martello and Paolo Toth. *Bin-packing problem*. 1990. URL: http://www.or.deis.unibo.it/kp/Chapter8.pdf (Accessed: April 23, 2016) (see p. 26).

[15] Michael P. Kassner. *Disaggregated data centers: great idea, but not just yet*. July 2015. URL: http://www.datacenterdynamics.com/servers-storage/disaggregated-data-centers-great-idea-but-not-just-yet/94473.fullarticle (Accessed: April 15, 2016).

[16] Ericsson and Intel Corporation. *Ericsson Introduces A Hyperscale Cloud Solution*. URL: http://www.intel.co.uk/content/dam/www/public/us/en/documents/solution-briefs/ericsson-hyperscale-cloud-solution-brief.pdf (Accessed: March 15, 2016).

[17] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. "Disaggregated memory for expansion and sharing in blade servers". In: *ACM SIGARCH Computer Architecture News*. Vol. 37. 3. ACM. 2009, pp. 267–278.

[18] Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parthasarathy Ranganathan, and Thomas F Wenisch. "System-level implications of disaggregated memory". In: *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*. IEEE. 2012, pp. 1–12.

[19] *Silicon photonics takes next step toward high bandwidth future*. 2015. URL: http://optics.org/news/6/3/24 (Accessed: April 13, 2016).

[20] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. "A scalable, commodity data center network architecture". In: *ACM SIGCOMM Computer Communication Review* 38.4 (2008), pp. 63–74.

[21] Andreas Wolke, Boldbaatar Tsend-Ayush, Carl Pfeiffer, and Martin Bichler. "More than bin packing: Dynamic resource allocation strategies in cloud data centers". In: *Information Systems* 52 (2015), pp. 83–95.

[22] Christoforos Kachris, Konstantinos Kanonakis, and Ioannis Tomkos. "Optical interconnection networks in data centers: recent trends and future challenges". In: *Communications Magazine, IEEE* 51.9 (2013), pp. 39–45.

[23] Frank David John Dunstan. "An algorithm for solving a resource allocation problem". In: *Operational Research Quarterly* (1977), pp. 839–851.

[24] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. “VL2: a scalable and flexible data center network”. In: *ACM SIGCOMM computer communication review*. Vol. 39. 4. ACM. 2009, pp. 51–62.

Appendices

A Source Code Reference

Given below is a list containing references to all the MATLAB source files that were developed and/or modified during the course of this project.

Source File	Author	Purpose/Modifications/Source
configType1.yaml	A. Tibrewal	Configuration file (Type 1)
configType2.yaml	A. Tibrewal	Configuration file (Type 2)
configType3.yaml	A. Tibrewal	Configuration file (Type 3)
BFS.m	A. Tibrewal	Contains customised breadth-first search algorithm
displayResults.m	A. Tibrewal	Used for generating and displaying results
inputGeneration.m	A. Tibrewal	Used for generating input requests
networkAllocation.m	A. Tibrewal	Used for allocating network resource
networkCreation.m	A. Tibrewal	Used for creating and initialising data centre graphs
plotDataCenterLayout.m	A. Tibrewal	Used for displaying data centre graphs
plotHeatMap.m	A. Tibrewal	Used for displaying data centre heat maps
plotUsage.m	A. Tibrewal	Used for displaying data centre usage dynamically
resourceAllocation.m	A. Tibrewal	Used for allocating IT resources
resultsGen.m	A. Tibrewal	Used for generating and displaying simulation results
runCompleteSimulation.m	A. Tibrewal	Used for initialising and managing the entire simulation

Continued on next page

Continued from previous page

simStart.m	A. Tibrewal	Used for starting the simulation for each type of data centre configuration
requestDB.mat	A. Tibrewal	Used for storing the request database
graphkshortestpaths.m	E. David Amir	Contains implementation of Yen's algorithm in MATLAB Source :`http://www.mathworks.com/matlabcentral/fileexchange/35397-k-shortest-paths-in-a-graph-represented-by-a-sparse-matrix--yen-s-algorithm-`
logb.m	B. Shoelson	Contains logarithmic base conversion function Source: `http://uk.mathworks.com/matlabcentral/fileexchange/14866-logb`
M-utilib library (Contains multiple source files)	E.Bezzeccheri	MATLAB utility library Source: `https://github.com/edobez/M-utilib`
subtightplot.m	F.G. Nievinski P. Kumpulainen S. Nikolay	Used for improving the layout of figures and plots/graphs Source: `http://www.mathworks.com/matlabcentral/fileexchange/39664-subtightplot`
yamlmatlab library (Contains multiple source files)	J. Cigler J. Siroky P. Tomasko	YAML parser for MATLAB Google Code Archive Source: `https://code.google.com/archive/p/yamlmatlab/`

Table 12: *List of source files developed and/or used through the course of this project.*

B Software/Tools Listing

Given below is the list of all the software tools that were used during the course of this project.

Software	Producer/Developer	Version	License
MATLAB	MathWorks	R2015b	Academic License
Git	Linus Torvalds	2.7.4	GNU General Public License v2

Table 13: *List of software/tools used through the course of this project.*