Multi-Cloud Application Deployment

by

Babneet Singh

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science (M.A.Sc.)

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

Deploying applications in a multi-cloud environment can be challenging. The cloud providers face the challenge of optimally utilizing the cloud resources to achieve a certain goal such as reducing the power consumption of the cloud. The cloud users, who deploy applications in the cloud, face the challenge of satisfying the application requirements, such as the processing and memory requirements, and application user requirements, such as the throughput and response time constraint. There is a relationship between the challenges faced by the cloud providers and cloud users, which allows these challenges to be studied together. This work covers such a problem, which aims to find a deployment, i.e. allocate resources for an application in a multi-cloud environment, while satisfying the above requirements of the application, application user and cloud provider. The proposed algorithm uses a combination of queueing theory, clustering, graph partitioning and bin packing strategies to solve the multi-cloud application deployment problem. It produces a solution within 20 seconds in 90% of the test scenarios, which is a reasonable amount of time to be useful in practice. In 77% of the test scenarios, the solution's power consumption is within 10% of an unachievable theoretical lower bound, showing that the heuristic algorithm is extremely effective.

Acknowledgements

I would like to express my profound gratitude to my supervisors, Professor C. Murray Woodside and Professor John W. Chinneck, for constantly guiding, inspiring and encouraging me during my thesis. This research would not be possible without their support and feedback. Also, it was a lot of fun to brainstorm with them. My most memorable moment was the discussion to name the algorithm using my initials, BS, or after McDonalds, McD ("billions served"). Unfortunately, I failed to provide strong arguments for such naming. So, we ended up naming the algorithm, MCAD.

Furthermore, I would like to thank my living family members, Rupdiner Kaur (mother) and Amarpreet Singh (brother), for their cooperation, and moral and emotional support. Finally, I would like to thank my father, Karamjit Singh, and my grandfather, Rajinder Singh, for providing strength in spirit through the lingering memories.

Table of Contents

List of Tables

List of Figures

Notation

Acronyms

2-D	Two Dimensional
BP	Bin Packing
FFDH	First-Fit Decreasing Height
FIFO	First in First out
FM	Fiduccia-Mattheyses
GB	Gigabytes
HASRUT	Heuristic Algorithm for Service-centre Resource UTilization
HBF	Hybrid Best-Fit
HOL	Head of Line
IaaS	Infrastructure as a Service
IoT	Internet of Things
JUNG	Java Universal Graph Framework
KL	Kernighan-Lin
LQN	Layered Queueing Network
LQNS	Layered Queueing Network Solver
MCAD	Multi-cloud Application Deployment
ms	Milliseconds
N/A	Not Available
NP	Non-deterministic Polynomial-time
PaaS	Platform as a Service
PM	Physical Machine
PPR	Preemptive Resume
PS	Processor Sharing
PUE	Power Usage Effectiveness
RUAEE	Resource Utilization Aware Energy Efficient
SaaS	Software as a Service
SLA	Service Level Agreement
SPEC	Standard Performance Evaluation Corporation
VM	Virtual Machine
W	Watts

Symbols

B_{cpu}	Upper bound on the processing requirement of a merged node during coarsening
B_{edge_factor}	Edge weight factor
B_{edge_weight}	Lower bound on the edge weight during coarsening
B_{mem}	Upper bound on the memory requirement of a merged node during coarsening
B_{node_count}	Lower bound on the number of nodes in the coarsened graph
C	Set of clouds
$C_{s\rightarrow mem}$	Constant factor to convert s_i into the memory requirement
$C_{s\rightarrow proc}$	Constant factor to convert s_i into the processing requirement
D	Partition of G with $\|C\|$ parts; also used for the deployment
E	Set of edges in a directed graph
$g_D(n, l)$	Gain from moving node n to cloud l in deployment D
G	Directed graph
ID_{delay}	Delay-sensitive initial deployment
ID_{power}	Power-sensitive initial deployment
$ID_{randomX}$	Random initial deployment with index X
M_i	Memory requirement of LQN entry i
M_{total}	Total memory requirement of an application
N_{bad}^{ID}	Number of initial deployments that fail to yield a solution
$N_{coarsen}^{nodes}$	Number of graph nodes with coarsening
N_{delay}^{ID}	Number of delay-sensitive initial deployments
N_{good}^{ID}	Number of initial deployments that yield a solution
$N_{no-coarsen}^{nodes}$	Number of graph nodes without coarsening
N_{power}^{ID}	Number of power-sensitive deployments
N_{random}^{ID}	Number of random initial deployments
$N_{replicas}$	Number of replicas for a graph node
N_{total}^{ID}	Total number of initial deployments
N_{LQNS}^{iter}	Number of iterations taken by the LQNS to produce a solution
P_{MCAD}	Power consumption of the solution deployment from the MCAD algorithm
P_{bound}	Theoretical lower bound on the power consumption
P_i	Processing requirement of LQN entry i in $ssj\ ops$
P_{max_moves}	Maximum number of moves to attempt on a node during partitioning
P_{total}	Total processing requirement of an application
P_{window}	Number of nodes to evaluate before a move during partitioning

R_{LQNS}	Response time per user request from the LQNS
R_{MCAD}	Response time per user request of the solution deployment from the MCAD algorithm
R_{MCAD}	Response time per user request of the solution deployment
R_{PT}	Total processing time of an application
R_{app}	Response time per user request for an application
$R_{constraint}$	Response time constraint per user request
$R_{net\_delay}(D)$	Total network delay for a deployment D
$R(D)$	Response time per user request of deployment D
s_i	CPU service demand per invocation of LQN entry i
s_t^w	Weighted average CPU service demand per invocation for LQN task t
$ssj\ ops$	Workload operations per second
T_{LQNS}	Time taken by the LQNS to solve an LQN model
T_{MCAD}	Time taken by the MCAD algorithm to produce a solution
U_{nom}	Nominal resource utilization
V	Set of nodes in a directed graph
X_i	CPU service demand per user request of LQN entry i in milliseconds
y_{ij}	Mean number of calls from LQN entry i to LQN entry j
Y_{ij}	Mean number of calls per user request from LQN entry i to LQN entry j
Y_j	Mean number of calls per user request to LQN entry j
Y_{total}	Total number of calls per user request in an application
Z_t	Think time of LQN task t
α_{wait}	Control parameter to account for the wait times and resource contention delay by adjusting the response time constraint
$\chi(D)$	Cut-weight or cut-size of a partition D
$\delta(r, q)$	Delay between two entities r and q in milliseconds
$\Delta_{\%}(A, B)$	Percentage difference between parameters A and B
$\gamma_j(x_i)$	Power consumption for assigning application component x_i to cloud j
$\omega(D)$	Power consumption of deployment D
φ	Application user parameter set
$\psi(D)$	Cut-set of a partition D
τ	Throughput requirement of an application user

List of Appendices

Chapter 1 Introduction

Cloud computing provides resources for computation: storage, servers, networking and software, over the internet [1]. There are challenges in efficiently deploying the applications on the clouds. From the cloud provider's perspective, a challenge is to reduce the operation costs while deploying the applications on the clouds. For an environment friendly cloud, there is the challenge of reducing the power consumption while deploying the application. From the application provider's perspective, a challenge is to satisfy the Service Level Agreement (SLA) requirements while deploying an application on the clouds. This work will explore a sub-set of these challenges and propose a solution to resolve them.

1.1 Applications: Monolithic versus Distributed

Applications can be generalized into two types: monolithic and distributed. Monolithic applications are deployed as a single process on a server where each application component "invokes another using a language-level call, such as a method or function" [2]. Such applications do not scale well in a cloud environment since they require vertical scaling, more powerful servers, to increase the throughput. In a cloud environment, the most loaded server becomes the bottleneck for monolithic applications.

A cloud provides access to multiple servers with different computing performance. In distributed applications such as a microservices-based application, each application component is "a process running in its own distinct network node that communicates with other components through an inter-process communication mechanism" [2]. These applications scale well in the cloud environment via horizontal scaling and replication. Most web-oriented systems are distributed, with multiple tiers

and geographic replication. Recent developments include container-based systems with many separate containers. There is also a trend of converting large monolithic enterprise applications into micro-services before deploying them on the cloud for better scaling. In summary, distributed applications are best suited for the cloud environments.

1.2 Cloud Services

There are four types of cloud services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Serverless computing, and Software as a Service (SaaS). IaaS corresponds to renting servers, virtual machines, storage, networks and operating systems from a cloud provider [1]. PaaS revolves around supplying an on-demand environment for developing, delivering, testing and managing software applications [1]. Serverless computing, which overlaps with PaaS, focuses on application development while the cloud provider manages the server management, setup and capacity planning. SaaS focusses on providing software applications over the internet on-demand [1]. PaaS, SaaS and Serverless Computing depend upon the cloud infrastructure or IaaS. A cloud provider, who is managing the cloud infrastructure and offering PaaS, SaaS and Serverless Computing, will face the challenge of efficiently utilizing the cloud infrastructure and reducing the operation costs. The same challenges will be faced by an IaaS user, who is responsible for deploying applications.

1.3 Multi-Cloud Architectures

Multi-cloud architectures are the future of cloud computing. Hybrid cloud computing uses a combination of multiple public or private clouds to perform certain tasks, with the orchestration of data between them [3]. Fog computing aims at fast cloud access by using a multi-tier cloud architecture where edge or small cloud nodes are

moved closer to where data is being generated. Edge computing is a solution derived from fog computing, where data analysis is moved closer to the edge nodes [4]. As the Internet of Things (IoT) expands, billions of devices will be connected to the internet. This will create a need for decentralized fast processing, decision making and data applications, which will require multi-cloud environments to operate efficiently.

For a software application, the SLA covers aspects such as security, reliability, cost, performance and support [5]. While deploying an application on the cloud, the main controllable SLA attribute is performance. Performance can be evaluated using metrics such as the throughput and response time [5]. These performance metrics are representative of the quality of an application's deployment in a cloud environment. The throughput states the number of user-requests an application handles per unit of time. The response time represents the time an application takes to process a user request. The application's performance model can be used to evaluate the performance characteristics while deploying the application in a cloud environment. A Layered Queueing Network (LQN) is an example of a performance model.

1.4 Real-time Applications

Real-time applications must satisfy a response time constraint in order to enhance user experience or satisfy user requirements [6]. Such applications will benefit from an algorithm that satisfies the response time constraint while deploying the application in a cloud environment. Examples of real-time applications are online gaming, video conference applications, Voice over Internet Protocol applications, storage solutions, e-commerce transactions, instant messaging, battlefield situational awareness applications, disaster response time applications, and real-time IoT [6]. Many of these applications are

distributed, have multiple modular application components, and have to handle large amounts of data while satisfying a response time constraint. The geographic location of computing resources plays a crucial role in satisfying the response time constraint because of the inter-cloud network delays. Multi-cloud environments will encompass clouds at different geographic locations, which will affect the latency constraints.

1.5 Problem Overview

It is desirable to deploy an application on multiple clouds since a single cloud may not have enough resources to accommodate the processing and memory requirements of an application. Also, an application may need to access multiple data sets that are available in different cloud locations. Furthermore, clouds will have servers with different computing characteristics and different network latencies that are dependent on the relative geographic location and network infrastructure. These cloud characteristics make it challenging to deploy an application in a multi-cloud environment. Furthermore, the performance constraints, such as the throughput and response time constraint, make it more difficult to efficiently deploy applications in a multi-cloud environment.

A deployment refers to the assignment of computing resources for satisfying the processing and memory requirement of an application. An efficient deployment refers to a deployment which reduces the cost or power consumption while satisfying the processing and memory requirements for an application. The main objective of this work is to find a power efficient deployment for an application in a multi-cloud environment while satisfying the throughput and response time constraint, the SLA requirements related to performance. The focus is on reducing the power consumption while deploying an application.

1.6 Thesis Contributions

This thesis presents a novel algorithm for deploying an application in a multi-cloud environment. The new algorithm is named the Multi-Cloud Application Deployment (MCAD) algorithm. The main contribution of the thesis is the MCAD algorithm, which is the only algorithm for multi-cloud application deployment that reduces the power consumption while satisfying the SLA requirements on the throughput requirement and response time constraint.

1.7 Thesis Organization

- Chapter 2 provides the background about performance modelling, LQN, cloud classification, power models of servers, graph partitioning and bin packing.
- Chapter 3 covers, the state of the art, the existing research work for deploying applications in a cloud environment.
- Chapter 4 contains the concise problem statement for deploying applications in a multi-cloud environment.
- Chapter 5 provides an overview for the MCAD algorithm. It also defines the environment and basic components in which MCAD operates.
- Chapter 6 contains the details about each individual component of the MCAD algorithm.
- Chapter 7 provides details about tuning the MCAD algorithm. It contains the setup, experiments, tuning criteria, data and analysis involved in tuning the MCAD algorithm.
- Chapter 8 contains the setup, experiments, validation criteria, data and analysis involved in validating the tuned MCAD algorithm.

- Chapter 9 covers the comparison between the MCAD and HASRUT [7] algorithms. The HASRUT algorithm is the base algorithm, which is used to develop the MCAD algorithm.
- Chapter 10 contains the conclusions, contributions and future work.

Chapter 2 Background

This section covers the concepts that are utilized in solving the multi-cloud application deployment problem. The concepts include Layered Queueing Networks, cloud classification, power models, graph theory, graph partitioning and bin packing.

2.1 Layered Queueing Network (LQN)

LQN combines a high-level model of software architecture with deployment and performance information. Its parameters capture the workload executed by the system in terms of CPU and I/O demands, and network messaging, and it can be solved as an extended queueing model to predict the system performance. As a performance model, it predicts the effect of congestion on both hardware and software resources. LQN modeling allows the study of throughput, response times, wait times, resource utilization and other performance characteristics of software systems. These performance characteristics are representative of real-life statistics.

An LQN model consists of four types of components: tasks, entries, calls and processors. A task represents a sequential program, which performs a set of operations for a queue of user requests. An entry is a sub-component of a task, and it represents one of the unique operations of the task. If we regard a task as a concurrent object, then its entries are its methods. A task can have multiple entries, which represent the different operations of the task. A processor is a hardware resource that is responsible for the physical execution of one or multiple tasks. A call represents a request for service from an entry of a task to an entry of another task. Further, a call can be categorized into three types: synchronous, asynchronous or forwarding.

The important parameters responsible for characterizing software workload in an

LQN model are the CPU demand of operations, think times, mean number of requests by an entry to another entry, and resource multiplicity. The CPU demand represents the time demand on the CPU, and it is a property of an entry. The think time represents any pure delay, where the processor is not utilized. Think times are used in modeling sources of workload such as users, and network latencies. The CPU service demand and think time can be associated with any unit of time if it is used consistently throughout the LQN model. Calls are made from one entry to another, with a parameter for the mean number of calls per invocation of the calling entry. Multiplicity represents the number of replicas of tasks or processors, and it is used to model the concepts of multi-threading and multi-processing.

Figure 20 shows an example of an LQN model in the graphical form. The same LQN model in text form is shown in Figure 21. A processor is represented with a circle in the graphical form and with the letter *"p"* in text form. The following information is associated with a processor: the processor name, the multiplicity, the scheduling discipline, an optional scheduling quantum and the speed factor. The speed factor reflects the speed of the processor in comparison to a model of processor chosen as a reference type. The reference type of processor is used to calibrate the CPU service demands of the entry operations, and the service demands on other processors are scaled by this speed factor. The processing discipline shows how the requests will be handled: first in first out (FIFO), processor-sharing (PS), random, priority preemptive resume (PPR), head-of-line priority (HOL). In the text view shown in Figure 21, "*p*" is followed by the processor name "*p2*", the PS quantity "*s 0.1*", the multiplicity "*m 1*" and the speed factor "*R 0.1*". In case a parameter is not specified, the default value is assumed. In the graphical view,

only the processor name "*c0*" and multiplicity "*{2}*" are shown for a processor.

In the graphical view shown in Figure 20, a task is represented with an outer parallelogram and an entry is represented with an inner parallelogram. An arrow has two purposes: it is used to associate a task and a processor, and it is also used to represent a call between two entries. In the text view, a task is represented with a letter "*t*", the think time is represented by the letter *"Z"*, the service demand of an entry is represented by the letter "*s*", and calls between the entries are represented by the letter "*y*". The following information is associated with a task: the task name, think time, multiplicity, list of entries and processor. The entry name and CPU service demand are associated with an entry. The mean number of requests, caller entry and callee entry are associated with a call. In the graphical view, the task name "*t2*", task multiplicity "*{2}*", entry name "*e2_0*", CPU service demand "*[1.96]*" and mean number of requests to another entry "*(1.14)*" are shown. In the text view, "*t*" is followed by the task name *"t2"*, list of entries "*n e2_0 -1*", name of the associated processor "*p2*", and the task multiplicity "*m 2*". "*s*" is followed by the entry name "*c0*" and CPU service demand "*1.37*". *"Z"* is followed by the entry name "*c0*" and think time "*3.5*". "*y*" is followed by the caller entry name "*c0*", callee entry name "*e0_0*" and mean number of requests "*1.86*". "*-1*" is used as a section or line terminator. This covers the minimal information required to understand LQN models in the graphical and text forms.

An outline of a sample solution for an LQN model, generated by the Layered Queueing Network Solver (LQNS), is shown in Figure 24 and Figure 25. Figure 24 contains the first half of the sample solution, and Figure 25 contains the second half of the sample solution. At the beginning, the solution shows the amount of time and

memory utilized by the LQNS to solve the LQN model. Then, the solution covers a lot of redundant information about tasks, entries and processors, which is provided in the LQN model. Then, the solution provides details about the delay, service time, throughput and utilization of each component in the LQN model. In the solution, the service time of the reference task, "*c0*" is the response time of the application.

2.2 Cloud Classification

Clouds can be classified based upon the size, latency, throughput and accessibility. Edge clouds are typically smaller in computation capacity, closer to the users (low latency) and larger in numbers. Central or regional clouds have larger computation capacity, are farther away from the users (high latency) and fewer in number. Figure 1 emphasizes cloud classification based upon the number of sites, footprint and power budget. Based upon accessibility, there are public and private clouds. Public clouds are completely managed by the cloud provider and accessible to everyone. Private clouds can be managed by an organization or a third-party provider. They are hosted on a private network and only accessible to the cloud owner.

Multi-cloud architectures comprise different kinds of clouds. The hybrid cloud is an example of a multi-cloud architecture, which is comprised of multiple public and private clouds. A public cloud provides access to cost-effective computation resources, and a private cloud provides the resources for handling sensitive data. Fog computing utilizes another multi-cloud architecture where the clouds are tiered based upon size. Figure 1 demonstrates a multi-cloud tiered architecture for supporting fog computing. The edge clouds support data collection and quick computations for the sensors, and the regional and central clouds support large-scale computations on the aggregate data

collected from all the sensors. Thus, there can be a sparse multi-cloud environment where an application can be deployed. [4] [3]

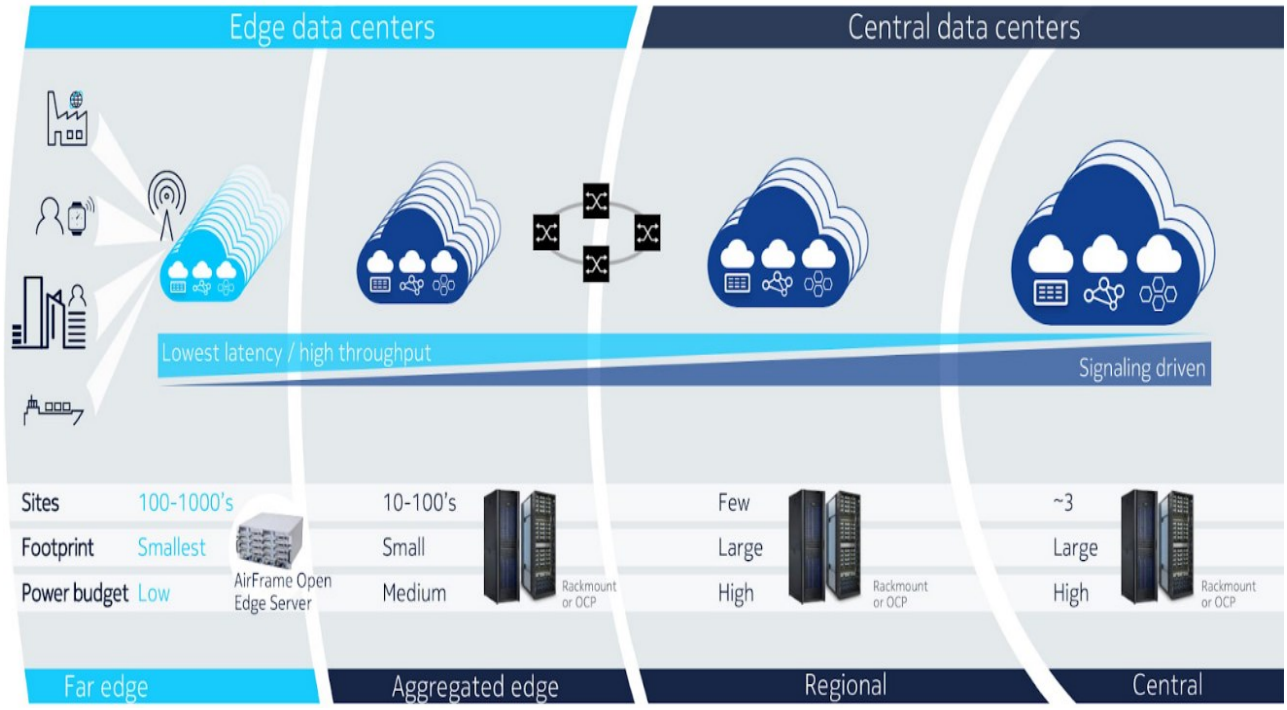

Figure 1 - Cloud Classification [4]

2.3 Power Models

A cloud can house servers with different power consumption characteristics. Efficiently deploying on the power efficient servers will lower the electricity costs of the clouds. The power consumption of servers can be either derived using existing mathematical models or empirical data. In [8], Möbius et al. provide a survey of the power consumption estimation models for the servers. The published results of the SPECpower benchmark [9] are a source of empirical data for the power consumption of the servers. Regression modelling can be applied to the empirical data in order to derive a power model for a server. An example of the derivation of a power model for a server is shown in Section 5.3.1.

2.4 Graph

A graph is an ordered pair: $G = (V, E)$, where V represents a set of nodes (vertices), and E represents a set of edges, which are two-element subsets of V i.e. an edge is related with two nodes. Directed graphs have a direction associated with the edges whereas undirected graphs have no direction associated with the edges. Weighted graphs have weights assigned with the nodes or the edges.

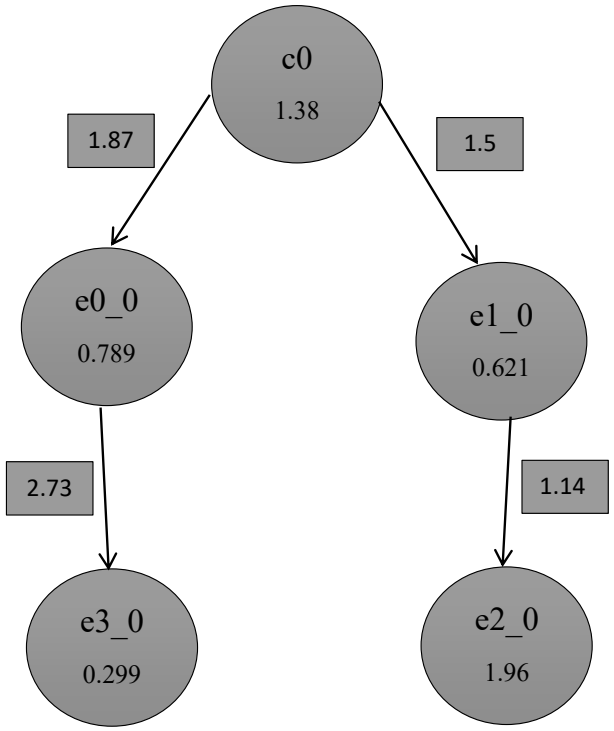

Figure 2 – A directed graph derived from the LQN model shown in Figure 20

Figure 2 shows an example of a weighted directed graph, which is derived from the LQN model shown in Figure 20. The nodes are the tasks from the LQN model: $V = \{c0, e0\_0, e1\_0, e2\_0, e3\_0\}$. The edges are the calls from the LQN model: $E = \{\{c0, e0\_0\}, \{c0, e1\_0\}, \{e0\_0, e3\_0\}, \{e1\_0, e2\_0\}\}$. The node weight represents the CPU service demand of the LQN task: $V = \{c0 \rightarrow 1.38, e0\_0 \rightarrow 0.789, e1\_0 \rightarrow 0.621, e2\_0 \rightarrow 1.96, e3\_0 \rightarrow 0.299\}$. The edge weight represents the mean number of

calls per invocation of the calling task: $E = \{\{c0, e0\_0\} \rightarrow 1.87, \{c0, e1\_0\} \rightarrow 1.5, \{e0\_0, e3\_0\} \rightarrow 2.73, \{e1\_0, e2\_0\} \rightarrow 1.14\}$.

2.5 Graph Partitioning

The purpose of graph partitioning techniques is to find a partition for a graph. For k-way graph partitioning, a partition consists of k parts, and a part is a subset of the graph's node set. The parts are non-intersecting and cover the entire node set of the graph. The most common objective of graph partitioning is to minimize the sum of the edge weights that connect nodes in separate parts of the partition. Balanced graph partitioning limits the total weight or size of each partition part. Imbalanced graph partitioning has no constraints on the total weight or size of each partition part. Graph partitioning is used in networking, scientific simulations, Very Large-Scale Integration circuits, and task scheduling in multi-processor systems [10].

The following is a mathematical representation of the k-way graph partitioning problem. Consider a directed graph $G = (V, E, f, h)$, where $f(x)$ and $h(\{x, y\})$ are weights of a node x and an edge $\{x, y\}$ respectively. Let the partition be a collection of the k subsets of V: $D = \{D_1, D_2, \dots, D_k\}$. Then, the cut-set of a partition is defined as $\psi(D) = \{\{x, y\} \in E \mid x \in D_m, y \in D_n, 1 \leq m < n \leq k\}$, and it contains the edges that connect nodes in separate parts of the partition. Furthermore, the cut-weight or cut-size of a partition is defined as

$$\chi(D) = \sum_{\{x,y\} \in \psi(D)} h(\{x, y\}) \tag{2.1}$$

The objective of the k-way graph partitioning problem with balance $1 \leq \alpha < k$ is to

$$\textbf{\textit{minimize}} \qquad \chi(P) \tag{2.2}$$

$$\textbf{\textit{subject to}} \qquad 0 < \sum_{x \in D_m} f(x) \leq \left\lfloor \frac{\alpha}{k} \sum_{y \in V} f(y) \right\rfloor, \forall m \in [1, k] \tag{2.3}$$

$$\bigcup_{1 \leq m \leq k} D_m = V \tag{2.4}$$

$$D_m \cap D_n = \emptyset \tag{2.5}$$

Equation 2.3 enforces the size limit on D_m set by α. Equations 2.4 and 2.5 enforce the definition of a partition.

Graph partitioning is an NP-hard problem [10]. So, it is difficult to find exact solutions for graph partitioning in a reasonable amount of time. Approximate solutions, that depend upon heuristics, work better in practice in terms of time. Menegola et al. [10] provides a survey of *k*-way graph partitioning techniques. The survey covers integer programs, heuristic techniques, multilevel techniques and multi-start techniques for graph partitioning.

1) For integer programs, the graph partitioning problem is translated into a mathematical model, which is a set of mathematical equations with an objective. Then, an integer program solver is used to solve the mathematical model, which yields an exact solution.
2) The heuristic based techniques yield an approximate solution, and they are divided into two categories: constructive and refinement algorithms. The constructive algorithms derive a start point or initial solution. The refinement algorithms (local search techniques) utilize heuristics to improve an existing

solution.

3) The multi-level techniques aim at reducing the size of a graph for the heuristic based techniques to yield improved results. The multilevel techniques have four main components: coarsening, initial partition, uncoarsening and refinement.
4) The multi-start techniques involve multiple initial partitions which are improved using local search techniques.

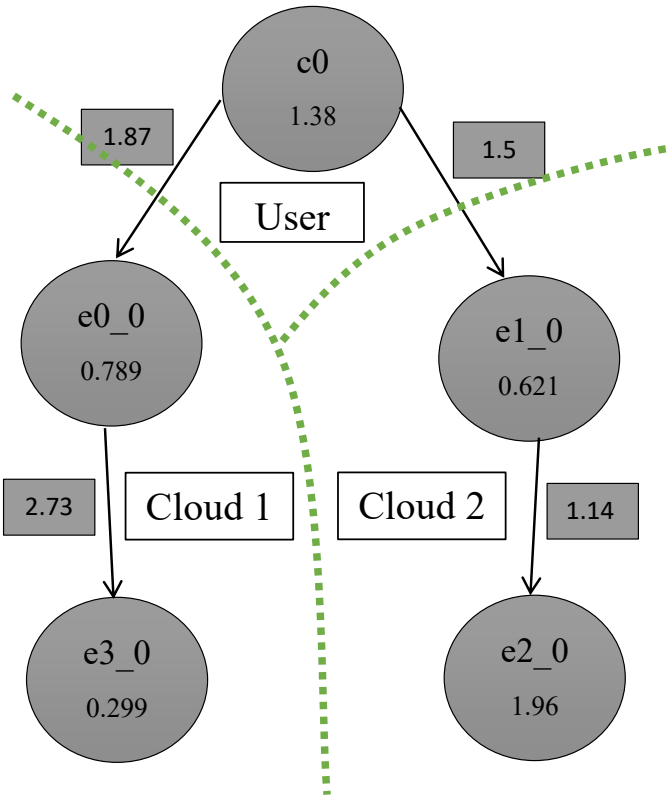

Figure 3 – Illustration of a three-way graph partition

Figure 3 provides an illustration of a three-way graph partition between the user, cloud 1 and cloud 2. The green dotted line represents the graph partition. The reference task, $c0$ is assigned to the user. The application components, $e0\_0$ and $e3\_0$, are assigned to cloud 1. The application components, $e1\_0$ and $e2\_0$, are assigned to cloud 2. The cut-weight is the sum of the edge weight of the following edges: $\{c0, e0\_0\}$ and $\{c0, e1\_0\}$,

which intersect the graph partition.

2.6 Fiduccia-Mattheyses (FM) Heuristic

The FM heuristic is a refinement heuristic for the graph partitioning problem. It is derived from the Kernighan-Lin (KL) heuristic. The KL heuristic has a time complexity of $O(n^2)$, and it is unable to change the imbalance of the partition in order to improve the cut-weight [10]. It should be noted that the KL heuristic swaps two nodes at a time between partitions. The FM heuristic decreases the time complexity to $O(m)$ in the best-case scenario by using a priority-queue based data structure, which handles insert, delete and update operations in $O(1)$ time [10]. It also handles imbalanced partitions by moving a single node at a time. The gain determines the change in cut-weight due to a move, and it is used as a metric for selecting the move to reduce the cut-weight. Moving a node, $n \in D_l$ to a graph part indexed by l' generates a new partition, D', where $n \in D'_{l'}$, and the gain is defined as

$$g_p(n, l') = \chi(D) - \chi(D') \tag{2.6}$$

where χ represents the cut-weight of the partition. Equation 2.1 shows the mathematical representation for the cut-weight. In simple words, the gain is the difference in the cut-weight between the old and new partitions. The cut-weight will reduce if a move with a positive gain is taken, and vice-versa.

2.7 Bin Packing

The goal of two-dimensional (2-D) bin packing is to pack a set of 2-D items into a minimum number of 2-D bins. It is an NP-hard problem [11]. It has applications in cutting stock, vehicle loading, pallet packing, memory allocation, and several other logistics and robotics related problems [11]. 2-D bin packing algorithm is well studied.

More details about 2-D bin packing approaches can be found in the following surveys: [11] and [12].

The following is a mathematical representation of the 2-D bin packing problem. Let the height be labelled as P, and width be labelled as M. Let $S(P, M)$ be used to evaluate the size of the bins and items. Consider a set of bins, $\{B_1, B_2, \dots\}$, with the same bin size $S(P_B, M_B)$, and a set of m items with sizes, $T = \{S(P_1, M_1), S(P_2, M_2), \dots, S(P_m, M_m)\}$. Equation 2.7 defines the objective function of 2-D bin packing, which is to find the minimum number of bins, N_{bins}, needed to pack the m items. Equation 2.9 ensures that the item sizes do not exceed the bin size. Equation 2.10 suggests that an item can only be placed in one bin.

$$\textbf{\textit{minimize}} \qquad N_{bins} = \sum_{i=1}^{m} v_i \tag{2.7}$$

$$\textbf{\textit{subject to}} \qquad N_{bins} \geq 1 \tag{2.8}$$

$$\sum_{j=1}^{m} S(P_j, M_j) u_{ij} \leq S(P_B, M_B) v_i, \forall i \in [1, n] \tag{2.9}$$

$$\sum_{i=1}^{m} u_{ij} = 1, \forall j \in [1, n] \tag{2.10}$$

$$v_i \in \{0, 1\}, \forall i \in [1, n] \tag{2.11}$$

$$u_{ij} \in \{0, 1\}, \forall i \in [1, n] \; \forall j \in [1, n] \tag{2.12}$$

where $v_i = 1$ if bin i is used, and $u_{ij} = 1$ if item j is packed into bin i.

The following three 2-D bin packing strategies are relevant to this work: First-Fit Decreasing Height (FFDH) algorithm, Hybrid Best-Fit (HBF) algorithm and a bin-pack approach derived from [13].

1. FFDH assigns the items, in decreasing height (first dimension), to the bin being

currently used. If a bin is unable to accommodate an application component, a new bin will be selected.

2. HBF is a two-phase algorithm. The first phase utilizes the best-fit decreasing height strategy, which sorts the items in decreasing order of the height. The second phase employs the best-fit decreasing strategy, which assigns an item to the bin with the least available height and width (second dimension) in which it will fit. Another variant is also employed where the height and width are interchanged; the items are sorted in decreasing order of the width in the first phase; and the second phase stays the same. The former HBF strategy is labelled as BP-HBF-Proc, and the latter HBF strategy is labelled as BP-HBF-Mem.
3. Han et al. [13] propose a server consolidation algorithm that reduces the number of physical machines (PMs) via live migration of virtual machines (VMs). In their server consolidation algorithm, a VM placement policy is proposed that assigns a VM to a PM during the migration process. The VM placement policy has characteristics of a bin packing heuristic. It assigns a VM to a PM such that the resource utilization of a PM is maximized. Ultimately, this should reduce the number of physical machines utilized. The VM placement policy depends upon the score calculation in [13], which evaluates whether a PM will have a balanced or unbalanced resource utilization before assigning a VM. A higher score corresponds to a balanced resource utilization, and a lower score corresponds to an unbalanced resource utilization. A balanced resource utilization suggests that more resources will be utilized on a PM. An unbalanced resource utilization suggests that less resources will be used on a PM. In [13], Equation 1 and

Equation 2 describe the score calculation, which is used as the basis for a bin packing approach. This bin-pack approach is labelled as BP-RUAEE.

Chapter 3 State of the Art

The literature review covers the research work related to deploying applications in the cloud that are relevant to the goals of this thesis, described next. A few keywords, related to deploying an application in the cloud, are resource allocation, server assignment, task scheduling, server consolidation, VM placement and resource provisioning. The existing application deployment strategies are evaluated based upon the following criteria:

1) Does it handle multiple heterogenous clouds i.e. clouds of different sizes?
2) Does it support heterogeneous servers i.e. servers with different processing and memory capacities?
3) Does it account for the network latencies?
4) Does it handle one application or a set of applications?
5) Does it handle distributed applications with communication between application components?
6) Does it satisfy the application's processing and memory requirements, and performance traits such as the throughput and response time constraint?
7) Does it have objective functions for power consumption, response time and other properties?
8) Is the solution time small enough to be useful in practice?

Molka and Casale [14] propose a new approach based on a hybrid genetic algorithm that efficiently handles resource allocation and server assignment for a given set of in-memory databases. The scenario is modelled as a multi-dimensional bin packing problem. The idea is to utilize non-linear optimizations to help the cloud providers

increase the energy-efficiency of in-memory database clusters in the cloud environments. This research considers response time, throughput and efficient energy consumption. But it only covers in-memory databases and single cloud environments. Also, it does not cover distributed scenarios where one database may communicate with another.

Tunc et al. [15] propose a task scheduling approach that satisfies the user's performance requirements while being energy efficient. A time dependent Value of Service metric is utilized to satisfy the response time constraint while accounting for the energy consumption. The tasks have an arrival time and a deadline, and they are assigned to VMs. A data driven energy model is used, which utilizes performance monitor counters to derive the power consumption of individual VMs. A Value-per-Total Resource metric is used to decide which VM to use for scheduling a task. This work only considers monolithic tasks. It does not account for multi-cloud task assignments. Also, it does not cover distributed scenarios where one task may communicate with another task in a multi-cloud environment.

Wang et al. [16] propose an energy optimal strategy for VM placement in the cloud. In their work, VMs are considered components of an application or independent tasks. A cubical power function is used to evaluate the power consumption of PMs: $P(f) = \alpha + \beta f^3$, where α is the static power consumption of the CPU, β is a constant coefficient, and f is the operating frequency of the CPU. In their solution, the processing and memory constraints are satisfied. The optimization is modelled as a mixed integer problem. This approach only accounts for a single cloud environment. It does not cover throughput and response time constraints. It only accounts for a set of monolithic applications. It does not cover distributed systems where tasks may communicate across

different clouds.

Frincu et al. [17] propose a multi-cloud resource provisioning algorithm that accounts for run-time cost, resource load, and application availability in case of failures. The applications are modelled with multiple components and connectors to represent distributed scenarios. The application components are scheduled on a set of VMs inside one or several clouds or even PMs in a plain-old data-center. The processing capacity, memory capacity and network load are accounted for in each VM or PM. The solution uses a genetic algorithm to satisfy multiple objectives when finding a deployment in a multi-cloud environment. The objectives are to maximize the resource usage, minimize the application run-time cost, and maximize the application availability and fault-tolerance. Genetic algorithms can be slow since multiple mutations of the initial population are explored to yield a solution. This work does not focus on energy efficiency while finding a deployment. It does not attempt to satisfy the overall response time of a distributed application. Also, the nodes, on which tasks are assigned, are homogenous i.e. they are of the same size.

Panda et al. [18] propose three task scheduling algorithms for a heterogeneous multi-cloud environment. The proposed task scheduling algorithms aim to minimize the makespan and maximize the average cloud utilization. Minimizing the makespan refers to minimizing the overall completion time needed to execute all the application tasks by the available clouds. This work does not consider either the throughput requirement or power consumption for scheduling the applications. Also, the cloud model does not account for individual resources on the cloud. The solution depends upon the cloud manager to handle the computation resources for the application.

Verba [19] proposes an application deployment framework for large-scale fog computing environments. The framework aims at solving the allocation problem of assigning highly connected applications to a set of gateways, which are devices with computing and networking capabilities. The framework uses graph clustering methods to solve the allocation problem while maximising the utility function. The utility function is decomposed into three components: the delay, reliability and constraint violations. But this work does not account for power consumption in the utility function.

Kaur [7] proposes the HASRUT algorithm for deploying a distributed application in a multi-cloud environment, which is the starting point for the present research. Her proposed solution uses a combination of multi-level, multi-start and local search graph partitioning techniques. It aims at minimizing power while satisfying the response time and throughput. It also satisfies the resource constraints on the processing and memory requirements of an application. The power model utilized is linear, and it depends upon the CPU utilization. But this work only considers an edge-core cloud configuration, which involves two clouds. So, this method does not work in a k cloud environment, where there can be any number of clouds. Also, the servers in the clouds are homogenous and identical i.e. every machine has the same processing and memory capacity. This does not reflect a real-life cloud where servers are heterogeneous and nonidentical with different processing, memory and power traits.

Chapter 4 Problem Statement

The problem is to develop an algorithm that finds a power efficient deployment for an application in a multi-cloud environment while satisfying the performance SLA requirements such as the throughput and response time constraint. The focus is on reducing the power consumption while deploying an application. Another main criterion is to produce a solution in a reasonable amount of time to be useful in practice. Chapter 3 lists the problem criteria and verifies that no solution has been proposed for this problem.

The following is a mathematical representation of the multi-cloud application deployment problem:

1. Let an application user be represented as $\varphi = \{\tau, R_{constraint}\}$, where τ is the throughput requirement and $R_{constraint}$ is the response time constraint.
2. Let an application be modelled as a directed graph, $G = (V, E, p, m, e, c)$ where V represents the set of application components, E represents the set of calls between the application components, $p(x)$ evaluates the processing requirement of an application component, $m(x)$ evaluates the memory requirement of an application component, $e(x)$ evaluates the total processing time of an application component, and $c(\{x, y\})$ evaluates the mean number of calls per invocation of the calling application component. It is assumed that the application components have been replicated in the directed graph to satisfy the throughput requirement, τ.
3. Let a cloud be represented as $C = \{\alpha, \beta, \gamma\}$, where α is the total processing capacity of the cloud, β is the total memory capacity of the cloud and $\gamma(x)$ evaluates the power consumption for assigning an application component to the cloud. When the application components are assigned to the clouds, it is assumed

that all application components are assigned to servers with sufficient processing and memory capacity to handle them. This assumption simplifies the problem model. In the solution, the cloud model contains a set of servers, γ is evaluated as a property of the servers on the cloud, and the application components are assigned to servers within the cloud.

4. Let the delay between two entities be evaluated using $\delta(r, q)$, where r and q can be either a cloud or user. It is assumed that the communication between servers in the same cloud has no delay.
5. For a multi-cloud environment, consider a set of k clouds, $\{C_1, C_2, \dots, C_k\}$. Let the deployment be a collection of the k subsets of V: $D = \{D_1, D_2, \dots, D_k\}$, where each subset represents the set of application components assigned to a cloud. Then, the cut-set is defined as $\psi = \{\{x, y\} \in E \mid x \in D_m, y \in D_n, 1 \leq m < n \leq k\}$. Let $f(x)$ evaluate to the entity which contains the application component. The entity can either be a cloud or user.
6. When the application components are deployed among the k clouds, then the total power consumption for the deployment is defined as

$$\omega(D) = \sum_{i=1}^{|V|} \sum_{j=1}^{k} u_{ij} \gamma_j(x_i) \tag{4.1}$$

where $u_{ij} = 1$ if application component x_i is assigned to cloud C_j. The total response time for the deployment is defined as

$$R(D) = \sum_{x \in V} e(x) + \sum_{\{x,y\} \in \psi} c(\{x, y\}) \delta(f(x), f(y)) \tag{4.2}$$

where the first component represents the total processing time (R_{PT}) of the application, and the second component represents the total network delay

(R_{net_delay}) due to inter-cloud communication. It is assumed that the delays within a cloud are negligible in comparison to the inter-cloud delays. The delays within a cloud are ignored in this problem model.

7. Equation 4.3 specifies the objective function of the multi-cloud application deployment problem, which is to minimize the power consumption of the deployment. Equation 4.4 ensures that the deployment satisfies the response time constraint. Equation 4.5 ensures that the total processing capacity of a cloud is not exceeded. Equation 4.6 ensures that the total memory capacity of a cloud is not exceeded. Equation 4.7 suggests that the union of the k subsets of the deployment form V. Equation 4.8 highlights that an application component can only be assigned to one cloud. In other words, there is no overlap between the deployment subsets.

$$\textbf{\textit{minimize}} \qquad \omega(D) \tag{4.3}$$

$$\textbf{\textit{subject to}} \qquad R(D) \leq R_{constraint} \tag{4.4}$$

$$\sum_{i=1}^{|V|} p(x_i)u_{ij} \leq \alpha_j, \forall j \in [1,k] \tag{4.5}$$

$$\sum_{i=1}^{|V|} m(x_i)u_{ij} \leq \beta_j, \forall j \in [1,k] \tag{4.6}$$

$$\bigcup_{1 \leq m \leq k} D_m = V \tag{4.7}$$

$$D_m \cap D_n = \emptyset \tag{4.8}$$

Chapter 5 The Multi-Cloud Application Deployment (MCAD) Algorithm: Overview

5.1 Outline

An overview of the MCAD algorithm is illustrated in Figure 4. The input parameters to the algorithm include information about the six key entities: clouds, servers, users, delays, application and control parameters. The details about the six key entities are provided in Section 5.3. The goal of the algorithm is to find a deployment for an application in a multi-cloud environment. The algorithm attempts to find a power efficient deployment that satisfies the processing, memory and throughput requirements, and the response time constraint for the application. It uses a combination of clustering, multi-level graph partitioning, multi-start graph partitioning, graph partitioning local search and bin packing techniques. The algorithm has nine different stages:

1. The first stage is to select the clouds for deploying the application. There will be many clouds close to the users. The clouds can be selected based upon characteristics such as the round-trip time to the user and size of the cloud. The goal is to reduce the complexity of the problem by reducing the number of clouds to consider in deploying an application.
2. The second stage is to generate a directed graph for the application using the application information. The directed graph captures the resource requirements and communication details between the application components and is needed for the partitioning stage. In this work, the application information is derived from the application's LQN model. The application information can be obtained from any other source until the directed graph for the application can be generated. The

second stage is discussed in more detail in Section 6.2.

1. INPUTS: clouds, servers, users, delays, application, control parameters
2. Select the clouds to deploy the application.
3. Generate a graph for the application.
4. Scale the application and generate a graph for the scaled application.
5. If coarsening is enabled, then:
6. Coarsen the scaled application and generate a graph of the coarsened application.
7. For $i \leftarrow 1$ to number of initial deployments:
8. If $i = 1$, generate a power-sensitive initial deployment.
9. Else if $i = 2$, generate a delay-sensitive initial deployment.
10. Else generate a random initial deployment.
11. If coarsening is enabled, then partition the coarsened application graph using the initial deployment.
12. Else partition the scaled application graph using the initial deployment.
13. If the partitioned deployment does not satisfy the response time constraint, then record the deployment and go to the next iteration in the loop.
14. If coarsening is enabled and bin pack before uncoarsening is enabled:
15. For each bin pack approach, perform bin packing on the partitioned deployment and record the deployment.
16. If coarsening is enabled, perform uncoarsening on the partitioned deployment and record the deployment.
17. For each bin pack approach, perform bin packing on the uncoarsened partitioned deployment and record the deployment.
18. OUTPUTS: the deployment that consumes the least power and satisfies the response time constraint or null if all deployments fail.

Figure 4 – Overview of the MCAD algorithm

3. The third stage is to scale the application and generate a directed graph of the scaled application. The application is scaled by addition of replicas in order to satisfy the throughput requirement. The scaling stage is discussed in more detail

in Section 6.3.

4. The fourth stage is to cluster the application components, also known as coarsening the scaled application's directed graph. The goal of coarsening is to reduce the graph size, which will lower the runtime effort for the MCAD algorithm in the subsequent stages. Coarsening is performed by merging two connected graph nodes into one graph node. This reduces the graph size and collapses the edge connecting the two graph nodes. The resulting merged node's resource requirements are an aggregate of the two source graph nodes' resource requirements. Another objective of coarsening is to avoid creating large merged nodes that cannot fit on any server. The coarsening stage is discussed in more detail in Section 6.4.
5. The fifth stage is to find an initial deployment, which is an initial k-way partition of the coarsened application graph. The initial deployment is a starting point for the partitioning stage, where the application components are assigned to different clouds based upon a flavor. In this work, there are three different flavors of initial deployment:
    i. The power-sensitive flavor is a greedy approach that assigns the application components to the most power efficient servers regardless of the cloud.
    ii. The delay-sensitive flavor is also a greedy approach, which assigns the application components either to the same cloud or the cloud closest to the previously assigned application component.
    iii. The random flavor assigns the application components randomly.

    In all the three initial deployment flavors, the resource requirements of the

application components are satisfied. The initial deployment stage is discussed in more detail in Section 6.5.

6. The sixth stage is to improve the initial deployment using the graph partitioning and bin packing techniques. The goal of partitioning is to satisfy the response time constraint by moving the application components from one cloud to another cloud, and achieve a power efficient deployment, using heuristics. The partitioning stage is discussed in greater detail in Section 6.6.
7. The seventh stage is to uncoarsen, which involves restoring the merged nodes into the original nodes in the partitioned deployment.
8. The eighth stage is to bin pack the application components into the servers on individual clouds. The three bin packing strategies utilized are BP-HBF-Proc, BP-HBF-Mem and BP-RUAEE (Section 2.7). The servers on the clouds are the 2-D bins, and the application components are the 2-D items used to fill the servers. The two dimensions are the processing and memory capacities. The processing capacity is considered as the height and the memory capacity is considered as the width of the 2-D bin. In this stage, the application components are not moved between clouds. Instead, they are just re-assigned to different servers in the same cloud in order to reduce the number of servers used for deploying the application. The goal is to reduce the power consumed by reducing the number of servers utilized for deploying the application. Applying the three bin pack strategies before and after uncoarsening yields six different deployments with different power characteristics. More details about the eighth stage are discussed in Section 6.8.

9. The fifth, sixth, seventh and eighth stages are repeated for each initial deployment, using one power-sensitive initial deployment, one delay-sensitive initial deployment, and a number of random initial deployments. In this work, the total number of initial deployments is chosen to be 52, with 50 random initial deployments. Each initial deployment yields seven candidates for the solution. The seventh stage provides the first candidate, and the eighth stage provides the remaining six candidates. With 52 initial deployments, there are $52 \times 7 = 364$ deployment candidates for the final solution. In the ninth stage, the deployment that consumes the least power and satisfies the response time constraint is chosen as the solution deployment.

Also, there is scope for concurrency since each initial deployment can be processed independently. So, stages five to eight can be run in a concurrent setting.

5.2 Notation

1. $N_{coarsen}^{nodes}$ represents the number of graph nodes with coarsening, and $N_{no-coarsen}^{nodes}$ represents the number of graph nodes without coarsening.
2. P_{total} represents the total processing requirement of the application in $ssj\ ops$; M_{total} represents the total memory requirement of the application in GB; and Y_{total} represents the total number of calls per user request in the application.
3. $N_{random}^{ID}\ (= 50)$ is the number of random initial deployments; $N_{power}^{ID}\ (= 1)$ is the number of power-sensitive initial deployments; $N_{delay}^{ID}(= 1)$ is the number of delay-sensitive initial deployments; and $N_{total}^{ID}\ (= 52)$ is the total number of initial deployments.
4. ID_{power} is used to label the power-sensitive initial deployment; ID_{delay} is used to

label the delay-sensitive initial deployment; and $ID_{randomX}$ is used to label a random initial deployment, where X represents the index of the random initial deployment.

5. The deployment from the seventh stage is labelled as $D_{partition}$.
6. In the eighth stage, the six variants of the three bin packing strategies are abbreviated as BP-HBF-Proc-Coarsen, BP-HBF-Mem-Coarsen, BP-RUAEE-Coarsen, BP-HBF-Proc-Uncoarsen, BP-HBF-Mem-Uncoarsen and BP-RUAEE-Uncoarsen.
7. The six deployments from the eight stage are labelled as $D_{BP\text{-}HBF\text{-}Proc\text{-}Coarsen}$, $D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$, $D_{BP\text{-}RUAEE\text{-}Coarsen}$, $D_{BP\text{-}HBF\text{-}Proc\text{-}Uncoarsen}$, $D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$ and $D_{BP\text{-}RUAEE\text{-}Uncoarsen}$.
8. N_{good}^{ID} represents the number of initial deployments that yield a solution; and N_{bad}^{ID} represents the number of initial deployments that fail to yield a solution.
9. R_{MCAD} represents the response time per user request of the solution deployment; P_{MCAD} represents the power consumption of the solution deployment; and T_{MCAD} represents the total runtime of the MCAD algorithm.

5.3 Environment and Basic Components

There are six key components involved in the MCAD algorithm: clouds, servers, users, delays, application and control parameters. The information about the six components is required to operate the MCAD algorithm. The subsequent sub-sections will discuss the six components in greater depth.

5.3.1 Server

A server represents a computing device with a processor, memory, motherboard,

power unit, cooling unit and network interface card. The basic purpose of the server is to perform computations. In this work, eight different types of servers are utilized. The servers differ in computing capacity, memory capacity and power characteristics. Thus, the servers are heterogeneous in nature.

In Appendix A, Table 21 provides a link to the SPECpower benchmark results for each server. The following information is obtained from the SPECpower benchmark results:

1. The number of hardware threads in the server. It represents the number of logical CPUs in the server. It will be used to track the usage of the processing capacity on each server.
2. The server memory size in GB. It will be used to track the usage of the memory on each server.
3. The SPECpower benchmark results use a term named $ssj\ ops$ to represent the processing capacity of the server. $ssj\ ops$ refers to the total number of operations finished during the benchmark's measurement interval divided by the interval time in seconds. It represents the SPECpower benchmark's throughput in terms of workload operations per second. The SPECpower benchmark results contain the $ssj\ ops$ for the following target loads: $100\%, 90\%, 80\%, \dots, 20\%, 10\%, 0\%$. The target load reflects the utilization of the server. The SPECpower benchmark results also contain the server's average active power in Watts (W) for the above target loads. A regression model is developed using the information about the $ssj\ ops$ and average active power. In this work, a polynomial regression model of degree three is used since it provides a smooth and accurate fit. In the regression model, the y-axis represents the average

active power and the x-axis represents the $ssj\ ops$. This regression model is used to evaluate the power for the specified $ssj\ ops$. It is used to evaluate the power when an application component is assigned to a server. Figure 5 shows the power versus throughput relationship for Fujitsu Server PRIMERGY TX1320 M2. The power-throughput relationship is shown as a polynomial regression model of degree three:

$$y = 2e^{-16}x^3 - 1e^{-10} + 7e^{-5}x + 12.184 \quad (5.1)$$

where y represents the power in W and x represents the throughput in $ssj\ ops$.

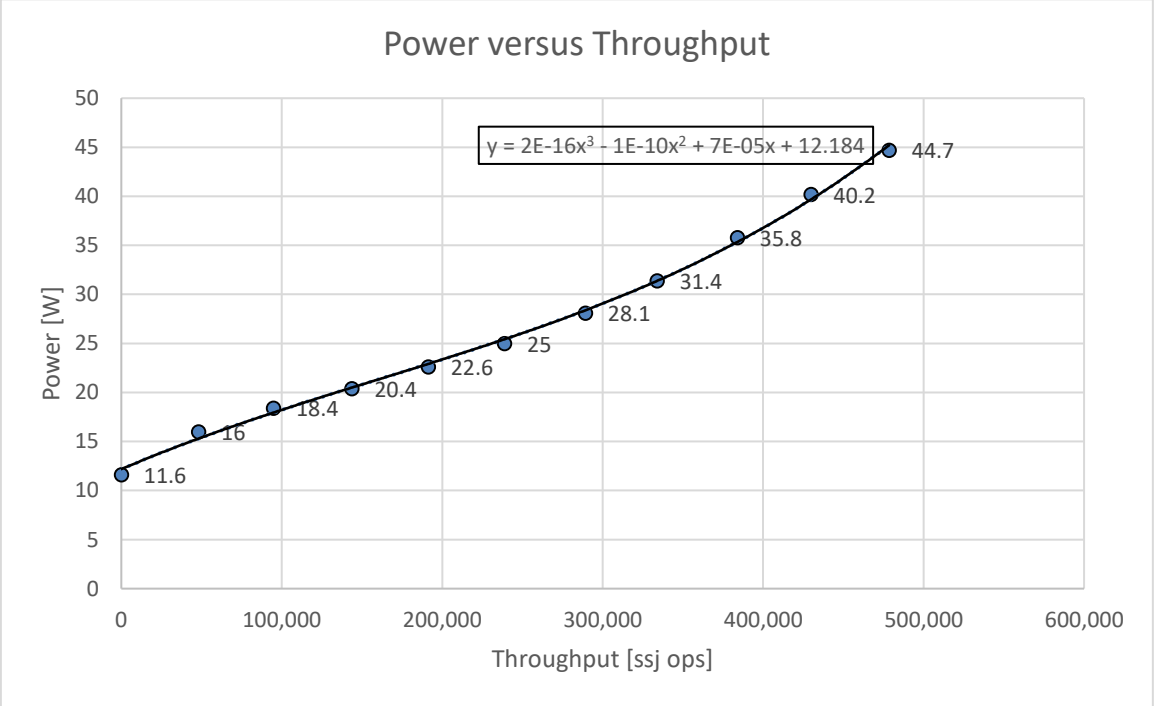

Figure 5 – Power versus throughput for Fujitsu Server PRIMERGY TX1320 M2

4. The overall score of the SPECpower benchmark represents the power efficiency of the server. It is the ratio of throughput and power, and its unit is $[(ssj\ ops)/W]$. The formal definition of the overall score is "the sum of the performance measured at each target load (in $ssj\ ops$) divided by the sum of the average active power (in W) at each target load including idle power" [20]. It will be used to sort the servers based upon

the power efficiency.

5.3.2 Cloud

A cloud represents a group of servers with a connecting internal network and peripherals. A cloud is distinguished by the amount of its computing resources. This work considers four types of clouds: the edge, small, medium and large, which are listed in the order of increasing size. The types and quantity of servers are varied in each cloud in order to create a set of heterogenous clouds. Also, the clouds are positioned such that the largest cloud has the greatest network delay to the user and smallest cloud has the least network delay to the user. In this work, the cloud environment is similar to a fog computing environment since it employs a multi-cloud architecture where the clouds are tiered based upon size.

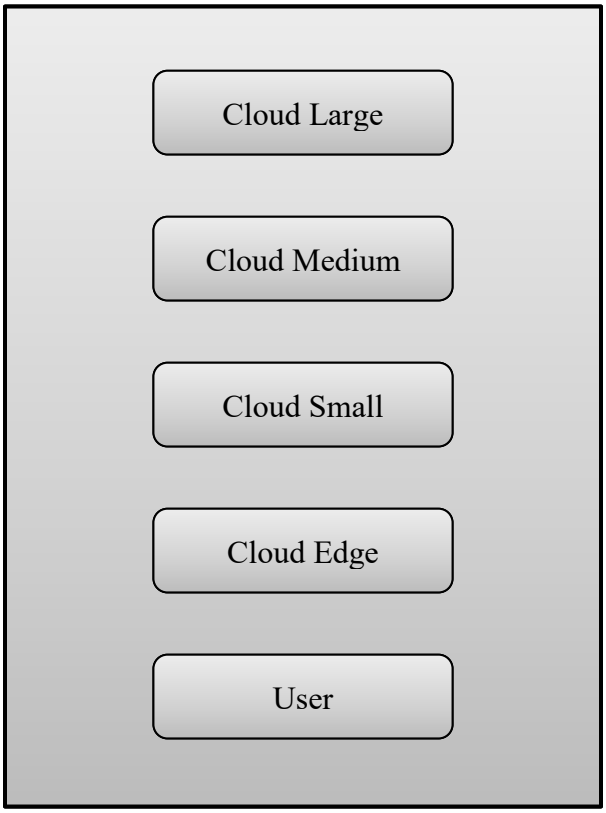

Figure 6 – Multi-cloud environment used in this work

In the experiments reported in this thesis, a four-cloud configuration is used as the assumed multi-cloud environment for evaluation. Figure 6 illustrates the four-cloud configuration along with the user. The details about the computing resources for these clouds are included in Appendix A. Table 16 summarizes the overall memory capacity in Gigabytes (GB), overall computing capacity in $ssj\ ops$ and number of CPUs for each cloud. Table 17, Table 18, Table 19 and Table 20 highlight the inventory of the servers in the edge, small, medium and large clouds respectively. The inventory represents the number of available servers and number of total servers of each kind. Table 21 lists the eight different kinds of servers used in this work.

In addition to the power consumed by a cloud's servers, the cloud's total power consumption includes the power losses due to inefficiencies, and power consumed by the cooling system and other elements, which are needed to support the operation of the cloud's servers. How to account for the cloud's total power consumption? In this work, a constant Power Usage Effectiveness (PUE) value is assumed across all the clouds. PUE is the ratio between the total power consumed by a cloud and power consumed by the cloud's servers [21]. The power consumed by a cloud's servers becomes a scaled down representation of the total power consumed by a cloud since a constant PUE value is assumed across all the clouds.

5.3.3 User

A user represents the consumer of the application to be deployed. For a group of users, the following two SLA requirements are specified: the throughput and response time constraint. The throughput is used to scale the application, and it is represented in user requests per milliseconds. The response time constraint specifies the upper bound on

the application's response time per user request, and it is represented in milliseconds.

5.3.4 Delays

A delay between two clouds is represented by the round-trip time of requests. It represents the network delay resulting from communication between two entities. In this work, there can be three different types of delays: a delay between a user and a cloud, a delay between two clouds and a delay within the same cloud. It is assumed that a delay within the same cloud is negligible in comparison to a delay between a user and a cloud or a delay between two clouds. So, the delays within the same cloud are ignored in the experiments. The MCAD algorithm can trivially be extended to model the delays within a cloud as a constant value that represents the average delay per request.

In Appendix A, Table 15 shows the delay between the entities of the evaluation cloud model, in milliseconds. Each row and column represents an entity. For example, column 2 represents the user and row 3 represents the edge cloud. The table cell, where column 2 and row 3 intersect, represents the delay between the user and the edge cloud. In this manner, the delay between other entities can be found. The chosen delay values are consistent with real-life network delays.

5.3.5 Application

The primary objective is to deploy distributed applications. This requires the following information regarding the components of a distributed application: the resource requirements and network delay due to communication between the application components. In this work, this information is derived from the application's LQN model. Other techniques can also be used to derive the application information. For example, the application can be profiled using performance analysis tools in order to obtain the

required information.

In the LQN model, s_e represents the CPU service demand per invocation for an entry e. Its unit is milliseconds. A call from one entry to another entry [22] is represented as an arrow labeled with the mean number of calls y_{de} from entry d to entry e. In this work, all calls are considered synchronous, which means that they implement remote procedure calls where the caller blocks to wait for a reply.

Before the application information is derived, the LQN model is simplified such that there is one entry per task and a maximum of one call between two entries. In the simplified LQN model, the single entry of a task represents an application component and a graph node. The operation of merging [22] entries of a task is defined as follows:
Let s_i be the CPU service demand of the merged entry i, and s_j be the CPU service demand of the original entry j of task t.

- Let y_{ik} be the mean number of calls from merged entry i to entry k of another task, and y_{jk} be the mean number of calls from original entry j to entry k.
- Let w_j be the fraction of calls to task t that go to entry j.

$$s_i = \sum_j w_j s_j \tag{5.2}$$

$$y_{ik} = \sum_j w_j y_{jk} \tag{5.3}$$

The above simplification process allows us to create an application graph as described in Section 2.4. The next step is to quantify the response time for the application from the simplified LQN model. Equation 5.4 displays that the application's response time R_{app} constitutes of two components. The first component R_{PT} represents the processing time of the application. The second component R_{net_delay} represents the

network delay due to communication between the application components. With respect to the problem statement (Chapter 4), Equation 5.4 for R_{app} is identical to Equation 4.2 for the total response time of a deployment.

$$R_{app} = R_{PT} + R_{net\_delay} \tag{5.4}$$

The network delay depends on the calls between the entries. In order to compute the response time per user request, the number of calls to each entry per user request is required. Equation 5.5 [22] shows the calculation for evaluating the number of calls to an entry per user request. It uses $Y_{user\_request} = 1$, where Y_e represents the number of calls to entry e per user request, and the sum is evaluated over all the entries that communicate with entry e. Equation 5.6 shows the calculation for evaluating the number of calls per user request between entry e and entry d, Y_{de}. Equation 5.7 defines R_{net_delay}, where δ_{de} represents the network delay per call between entry d and entry e, and ψ represents the set of entry pairs that correspond to the edges in the cut-set of a partition (deployment).

$$Y_e = \sum_d Y_d y_{de} \tag{5.5}$$

$$Y_{de} = Y_d y_{de} \tag{5.6}$$

$$R_{net\_delay} = \sum_{\{d,e\} \in \psi} Y_{de} \delta_{de} \tag{5.7}$$

A full solution for the wait times and resource contention delay from a performance model can be very expensive. A different approach is taken for approximating the wait times and resource contention delay in the MCAD algorithm. It is assumed that a good deployment will utilize the servers up to a certain maximum U_{nom}, which represents the nominal resource utilization of a server. It is further assumed, as a worst case, that all servers will be loaded to U_{nom}; although, this will not always be the

case. The values of U_{nom} are restricted to the following interval: $(0,1)$. A simple approximation, for the wait times and resource contention delay, based upon the queues with the PS discipline (an approximation to the time-slicing scheduling) relates the response time to the CPU service demand, as shown in Equations 5.8 and 5.9. Equation 5.8 defines X_d, which is the total processing time per user request for an entry d. Equation 5.9 defines R_{PT}, where the sum is evaluated over all the entries of the simplified LQN model. In Equation 5.9, the wait times and resource contention delay are accounted by inflating the processing time using the multiplication factor of $[1/(1-U_{nom})]$, and the slowest server speed is used.

$$X_d = Y_d s_d \tag{5.8}$$

$$R_{PT} = \sum_d \frac{X_d}{1-U_{nom}} \tag{5.9}$$

The subsequent step is to quantify the computation resources for the application from the simplified LQN model. In practice, the memory requirement is set by the application provider, and it is not related s_i. For simplicity, s_i is used to derive the processing requirement (in $ssj\ ops$) and memory requirement (in GB) for a merged entry in this work. The conversion is performed using constant factors. $C_{s\to proc}$ represents the constant factor to convert s_i into the processing requirement. $C_{s\to mem}$ represents the constant factor to convert s_i into the memory requirement. Equation 5.10 shows the calculation for evaluating the processing requirement, where P_i represents the processing requirement of a merged entry i. Equation 5.11 shows the calculation for evaluating the memory requirement, where M_i represents the memory requirement of a merged entry i. In Equation 5.10, the multiplication factor of $(1/U_{nom})$ inflates the processing requirement of a merged entry and prevents the server load to exceed U_{nom} in order to

enforce the simple approximation for the wait times and resource contention delay.

$$P_i = \frac{C_{s\rightarrow proc} s_i}{U_{nom}} \tag{5.10}$$

$$M_i = C_{s\rightarrow mem} s_i \tag{5.11}$$

$C_{s\rightarrow proc}$ and $C_{s\rightarrow mem}$ can be derived experimentally using benchmarks. In this work, $C_{s\rightarrow proc}$ and $C_{s\rightarrow mem}$ are chosen such that they provide realistic values for the processing and memory requirements. $C_{s\rightarrow mem}$ is chosen such that the memory requirement ranges between 256 MB to 4 GB for the application components. The average throughput per CPU in $ssj\ ops$ ranges between $40{,}000\ to\ 50{,}000$. $C_{s\rightarrow proc}$ is chosen such that the processing requirement ranges between 1 CPU to 4 CPUs for the application components. In this work, the smallest server has 8 CPUs and 16 GB of memory. So, the above evaluation for the processing and memory requirements assures that the application components will fit on to the smallest and slowest servers.

5.3.6 Control Parameters

The control parameters determine how the MCAD algorithm will run. An example of a control parameter is the parameter to enable or disable coarsening in the algorithm. The control parameters specified in all the sections are managed by the user of the MCAD algorithm.

Chapter 6 MCAD Algorithm Details

In this chapter, the MCAD algorithm, introduced in Chapter 5, is discussed in greater detail. The theory of the different algorithm stages is provided in the subsequent sections.

6.1 Stage 1: Cloud Selection

This algorithm supports k clouds. The algorithm does not include a cloud selection process. A dynamic approach to select clouds will be useful in real life scenarios where many clouds might be used. When deploying distributed applications in a multi-cloud environment, the cloud selection process will reduce the number of clouds involved in the problem; thus, lowering the runtime effort of the algorithm. Cloud properties, such as the available resources and delays between other clouds and users, can be used in the cloud selection process. Also, the distributed application's resource requirements can be used as a decision-making factor in the cloud selection process.

6.2 Stage 2: LQN Model to Application Graph Transformation

This section describes the second stage of the algorithm in greater detail. The input LQN model is parsed, simplified and converted into a directed graph. Figure 7 shows the algorithm for this process.

All entries of the task are aggregated and simplified into one entry. Each task ends up having one entry, which represents the weighted average CPU service demand per invocation (s^w) of all the entries in a task. Equation 5.2 shows the calculation of s^w . Each task is converted into a node in the directed graph. The node inherits s^w as a property. This property is used to evaluate the processing and memory requirement for the graph node. More details regarding the conversion from CPU service demand per

invocation to processing and memory requirement can be found in Section 5.3.5.

1. INPUTS: LQN model
2. Instantiate a directed graph with no edges and no vertices.
3. For each LQN task in the LQN model:
4. Merge the LQN entries in the LQN task into one LQN entry with s^w.
5. Add the merged LQN entry as a node in the directed graph.
6. For each LQN call in the LQN model:
7. If an edge already exists between the source LQN task and the destination LQN task in the directed graph then:
8. Increment edge weight by LQN call's y value.
9. Else add a new edge in the directed graph with weight set to LQN call's y value.
10. Recursively evaluate Y for each edge in the directed graph.
11. Evaluate X for each node in the directed graph.
12. OUTPUTS: the directed graph on success and null on failure.

Figure 7 – Algorithm for generating a directed graph for the application

In an LQN model, a call is represented by an arrow between two entries. The arrow is labelled with the mean number of calls between the two entries. Since the entries in a task have been aggregated to one entry per task, all calls between two tasks are aggregated to represent one call. The aggregated call is introduced as an edge in the directed graph. The graph edge inherits the aggregated mean number of calls per user request (Y) as a property. This property is evaluated using Equations 5.5 and 5.6, and it is used to calculate the network delay between two nodes if they are deployed on different clouds. The CPU service demand per user request (X) for each node is calculated using Equation 5.8. From the application graph, the total processing time and delay for the application can be calculated using Equations 5.7 and 5.9 respectively.

6.3 Stage 3: Scaling the Application

This section provides the details about the third stage of the algorithm, where the application is scaled horizontally by the addition of the replicas, to provide enough capacity to satisfy the throughput requirement.

For a node, the number of replicas, $N_{replicas}$, is evaluated using Equation 6.1, where s^w is the weighted average CPU service demand per invocation, Y is the mean number of calls per user request, τ is the throughput requirement in user requests per second and U_{nom} is the nominal resource utilization. τ is specified as part of the SLA requirements. More details about the throughput can be found in Section 5.3.3, and U_{nom} can be found in Section 5.3.5.

$$N_{replicas} = \left\lceil \frac{(\tau s^w Y)}{U_{nom}} \right\rceil \tag{6.1}$$

How is a distributed application scaled? Figure 8 sheds light onto the scaling process. The application graph and throughput are used to scale the distributed application. If replication is required, then the output is a new directed graph, which represents the scaled application. If replication is not needed, then the application graph is used in the subsequent stages of the algorithm.

In the first stage of scaling, the replicas are created for the nodes in the application directed graph using Equation 6.1. The original and replica nodes are added to the new directed graph. The replica nodes inherit the weighted average CPU service demand per invocation (s^w) from the original node. In the second stage of scaling, each edge to or from an original node is replaced by an edge to or from every replica, representing the division of traffic among the replicas. The mean number of calls per user request (Y) is distributed equally among the replicas. This uniformly divides the weighted average CPU

demand per user request among the replicas.

The processing and memory requirements for a replica are evaluated using Equations 5.10 and 5.11, which guarantee that a replica will fit on the smallest and slowest server.

1. INPUTS: application directed graph, throughput, U_{nom}
2. If $N_{replicas}$ is not greater than 1 for any node in the application directed graph, then scaling is not performed.
3. Instantiate a scaled directed graph with no edges and no vertices.
4. For each node in the application directed graph:
5. Evaluate $N_{replicas}$ for the node.
6. Add the node and its replicas as nodes in the scaled directed graph.
7. For each edge in the original directed graph:
8. Evaluate $N_{replicas}$ for the edge's destination node.
9. New edge weight ← current edge weight / $N_{replicas}$.
10. For each instance of the edge's source node in the scaled directed graph:
11. For each instance of the edge's destination node in the scaled directed graph:
12. Using the new edge weight, add a new edge between the instance of source node and the instance of destination node in the scaled directed graph.
13. OUTPUTS: If scaling is successfully performed, then the scaled directed graph is returned. Otherwise, the input application directed graph is returned.

Figure 8 – Algorithm to generate a directed graph for the scaled application

6.4 Stage 4: Coarsening

This section contains the details about coarsening the application graph, which is the fourth stage of the algorithm. For a distributed application, coarsening is analogous to combining the resource requirements of multiple application components. Coarsening involves merging of nodes in the graph. As a result of merging nodes, the edge between

the two merged nodes disappears. Thus, the merged nodes will be deployed together on the same server in a cloud. This helps to avoid the delay if the merged nodes were deployed on different servers and different clouds.

Figure 9 contains the algorithm for the coarsening stage. The input to the coarsening stage is a directed graph, of either the scaled or original application, and control parameters. If coarsening is performed, then the output is a new directed graph that represents the coarsened application. Otherwise, the input directed graph is used in the subsequent stages of the algorithm. For the coarsening stage, there are four control parameters, which are used to manage the granularity of the coarsened directed graph.

During the coarsening stage, an upper bound is imposed on the processing and the memory requirements of a merged node. This makes the common deployment of the cluster of tasks feasible. B_{cpu} and B_{mem} are used to represent the upper bound on the processing and memory requirements of a merged node respectively. B_{cpu} and B_{mem} are evaluated using the processing and memory capacity of the servers such that a merged node can be assigned to any server. B_{cpu} and B_{mem} are categorized as control parameters. There are outlier nodes whose processing and memory requirement exceeds the above-mentioned upper bounds. Currently, the outlier nodes are exempted from being merged.

There are other control parameters for the coarsening stage. Minimum node count, B_{node_count} is a control parameter used to specify a lower bound on the number of nodes in the coarsened graph. Also, coarsening will not be performed if the input directed graph has fewer nodes than B_{node_count}. The purpose of B_{node_count} is to prevent all the nodes from being merged into one node. Edge weight factor, B_{edge_factor} is a control parameter used to derive a lower bound on the edge weight, B_{edge_weight}. The edge

weight corresponds to the mean number of calls per user request. B_{edge_factor} is used as a constant factor. It is multiplied with the average edge weight of the scaled directed graph to derive B_{edge_weight}. Only the edges which have an edge weight greater than B_{edge_weight} are collapsed.

Multi-level coarsening is allowed, which means that the merged nodes can be further merged with other nodes to form a new merged node. For each merge operation, a record is created, which contains the information about the two merged nodes. These records are used to restore the merged nodes into the corresponding original nodes in the uncoarsening stage.

1. INPUTS: scaled directed graph, B_{node_count}, B_{edge_weight}, B_{cpu}, B_{mem}, number of nodes in the scaled directed graph
2. If the number of nodes in the scaled directed graph is less than B_{node_count}, then coarsening is not performed.
3. Instantiate a coarsened directed graph with no edges and no nodes.
4. Sort the edges in the scaled directed graph in decreasing order of edge weight.
5. $B_{edge\_weight} \leftarrow B_{edge\_factor}$ × average edge weight of the scaled directed graph.
6. For each edge in the sorted set:
7. If (edge weight > B_{edge_weight}) and
   (number of nodes in scaled directed graph > B_{node_count}) then:
8. Merge the source and destination nodes.
9. If the merged processing and memory requirement is less than B_{CPU} and B_{mem}, then:
10. Add the merged node to the coarsened graph.
11. Decrement number of nodes by 1 and go to the next iteration in the loop.
12. Add the source and destination nodes to the coarsened directed graph if they are not merged.
13. If the edge does not exist in the coarsened directed graph, then add it.
14. Else increment the number of calls per user request (Y) for the merged edge.
15. OUTPUTS: If coarsening is successfully performed, then the directed graph of the coarsened application is returned. Otherwise, the input directed graph of the scaled application is returned.

Figure 9 – Algorithm to generate a directed graph after coarsening

6.5 Stage 5: Initial Deployments

This section provides the details about the initial deployments of the algorithm. The goal is to assign the nodes to the servers on the clouds in order to get a starting point for the partitioning stage. The initial deployments are neither required to be power efficient nor satisfy the response time constraint. The following three initial deployment strategies are discussed in more detail below: deploy on the power efficient servers

(power-sensitive), deploy on the next closest cloud (delay-sensitive) and deploy randomly on clouds (random). The first and second strategy will each yield one starting point whereas the third strategy will yield multiple starting points.

All three initial deployment strategies take a directed graph and a list of clouds as the input. The output is a deployment, which contains a list of entries for each cloud. An entry contains the following information: the node, server and power consumed. Whenever a node is assigned to a cloud, a server is allocated, and the power consumption is evaluated for the node. The server is allocated such that the processing (P_{node}) and memory (M_{node}) requirements of the node are satisfied. P_{node} and M_{node} are evaluated using Equations 5.10 and 5.11 respectively. The power consumption is evaluated using the server's power model (Section 5.3.1).

6.5.1 Power-sensitive Initial Deployment

1. INPUTS: directed graph, clouds
2. Instantiate a deployment with no elements.
3. Sort the nodes in the directed graph in decreasing order of processing requirement.
4. For each node in the sorted set:
5. Find the cloud which requires the least power to allocate the node.
6. If a cloud is found, then add a deployment entry {node, server, power} to the list associated to the cloud.
7. Else fail since no cloud with enough resources was found to allocate the node.
8. OUTPUTS: the deployment on success and null on failure.

Figure 10 – Algorithm to deploy on the power efficient clouds first (power-sensitive flavor)

Figure 10 covers the power-sensitive initial deployment strategy, which assigns the nodes to the most power efficient servers among all the clouds. With this strategy, a power efficient initial deployment is attained. This deployment's power consumption will

not represent the theoretical lower bound for power consumption because resource fragmentation will prevent servers from operating at their optimal power efficient utilization. Resource fragmentation refers to the unused processing and memory capacities of a server. Also, the response time constraint may not be satisfied if the most power efficient servers are on the cloud furthest away from the user. The overall score of the SPECpower benchmark, described in Section 5.3.1, is used to sort the servers based upon the power efficiency.

6.5.2 Delay-sensitive Initial Deployment

1. INPUTS: directed graph, clouds
2. Instantiate a deployment with no entries.
3. chosen_cloud ← null.
4. Sort the nodes in the directed graph in decreasing order of processing requirement.
5. For each node in the sorted set:
6. If chosen_cloud is null, then:
7. Select the cloud with least delay to user, which can allocate the node, as chosen_cloud.
8. Else if chosen_cloud cannot accommodate the node then:
9. Find the cloud with least delay to chosen_cloud that can allocate the node, and update chosen_cloud.
10. If chosen_cloud is not set, then fail since no cloud with enough resources was found to allocate the node.
11. Else add a deployment entry {node, server, power} to the list associated to the chosen_cloud.
12. OUTPUTS: the deployment on success and null on failure.

Figure 11 – Algorithm to deploy on the closest clouds (delay-sensitive flavor)

Figure 11 covers the delay-sensitive initial deployment strategy, which assigns the nodes to the cloud closest to the current cloud if no resources are available on the current cloud. The first cloud chosen is the closest to the user. In the current context, closeness

represents the delay between two clouds. This strategy aims to satisfy the response time constraint. On the chosen cloud, the nodes are assigned to the power efficient servers. But the deployment's power consumption may not be power efficient if the closest clouds do not have the most power efficient servers.

6.5.3 Random Initial Deployment

Figure 12 covers the random initial deployment strategy, which randomly assigns the nodes to the clouds. On the chosen cloud, the nodes are assigned to the power efficient servers. With this strategy, the initial deployment may neither satisfy the response time constraint nor be power efficient. But the randomness will yield starting points that may result in the best solution deployment at the end of the partitioning stage.

1. INPUTS: directed graph, clouds
2. Instantiate a deployment with no entries.
3. Shuffle list of nodes in the directed graph to generate a randomly ordered list.
4. For each node in the random set:
5. Find a cloud randomly that can allocate the node.
6. If a cloud is found, then add a deployment entry {node, server, power} to the list associated to the cloud.
7. Else fail since no cloud with enough resources was found to allocate the node.
8. OUTPUTS: the deployment on success and null on failure.

Figure 12 – Algorithm to deploy randomly on the clouds (random flavor)

6.6 Stage 6: Partitioning

This section describes the sixth stage of the algorithm. The objective of the partitioning stage is to derive a power efficient deployment that satisfies the response time constraint. This is achieved by moving the nodes from one cloud to another cloud using heuristics. Kaur [7] achieved the best results using the FM heuristic (Section 2.6) while solving the edge-core (two-way) cloud deployment problem. So, a k-way variant of

the FM heuristic is utilized in the partitioning stage. Furthermore, the application components are assigned to servers on a cloud using the FFDH bin packing approach (Section 2.7) since it serves as a fast heuristic for server allocation and produces a power efficient deployment.

The partitioning stage has four main components: the stop condition (Figure 13), total network delay (R_{net_delay}) calculation (Figure 14), gain calculation (Figure 15) and server selection process (Figure 16). Finally, Figure 17 shows the entire partitioning stage.

Figure 13 describes the stop condition. The stop condition determines the termination point for the partitioning stage. If the partitioning stage has tried moving each node once and is unable to move any nodes, then the partitioning stage is not making any progress. This is considered as one of the termination points. The second termination point is determined by the user of the MCAD algorithm, who can define the maximum moves per node (P_{max_moves}), which represents the maximum number of times an attempt will be made to move a node. For large values of P_{max_moves}, the partitioning phase may yield a better solution. But, the runtime of the partitioning phase will also increase. If either of the termination expressions yield true, then the partitioning phase will terminate.

Figure 14 describes the calculation for R_{net_delay}, which represents the total inter-cloud network delay per user response for the deployment. Once the nodes are assigned to different clouds, there will be inter-cloud edges. Their edge weight will represent the number of calls per user response to a node. R_{net_delay} is the total network delay resulting from these calls. In this work, R_{net_delay} is represented in milliseconds.

Equation 5.7 provides a formal mathematical definition of R_{net_delay} in the context of this work.

1. INPUTS: $P_{max\ \_moves}$, total number of nodes, attempted tries, moves taken
2. If the moves taken is 0 after evaluating all the nodes, then stop partitioning.
3. If the number of attempted tries has exceeded the product of $P_{max\ \_moves}$ and the total number of nodes, then stop partitioning.
4. OUTPUTS: True to stop partitioning and false to continue partitioning.

Figure 13 – Algorithm for the stop condition

1. INPUTS: directed graph, partition, root node, user
2. Form a set of all edges which either (a) originate in the root node, or (b) have source and destination nodes in different clouds.
3. $R_{net\ \_delay}$ is the sum, over all edges in this set, of the products of calls and delay.
4. OUTPUTS: $R_{net\ \_delay}$ on success and a negative value on failure.

Figure 14 – Algorithm to calculate the total network delay (R_{net_delay})

1. INPUTS: node in cloud1, cloud1, cloud2, directed graph, partition, root node, user
2. Find $R_{net\ \_delay}$ before the move where the node stays in cloud1.
3. Find $R_{net\ \_delay}$ after the move assuming the node is moved from cloud1 to cloud2.
4. Gain ← $R_{net\ \_delay}$ before the move – $R_{net\ \_delay}$ after the move.
5. OUTPUTS: the gain on success and a negative value on failure.

Figure 15 – Algorithm to calculate the gain for moving a task

Figure 15 describes the gain calculation for moving a node from one cloud to another cloud. The gain is the change in R_{net_delay} due to moving a node from one cloud to another cloud. The gain is positive if a move reduces R_{net_delay}; and the gain is negative if a move increases R_{net_delay}. If the gain is zero then, there will be no change in R_{net_delay} if a move is taken. The unit of gain is milliseconds. Equation 2.6 provides a formal definition of the gain in the context of graph partitioning.

Figure 16 describes the process of choosing a server on a cloud for a node. The node and a list of servers are used as input. The servers are sorted in decreasing order of the power efficiency using the overall score of the SPECpower benchmark, which is described in Section 5.3.1. The application components are sorted in decreasing order of the processing requirement. The first server to satisfy the processing and memory requirements of the node is selected. This approach of assigning the nodes to servers is equivalent to the FFDH bin packing strategy, which is described in Section 2.7. P_{node} and M_{node} are evaluated using Equations 5.10 and 5.11 respectively. Refer to Section 5.3.5 for details about $C_{s\rightarrow proc}$, $C_{s\rightarrow mem}$ and U_{nom}.

1. INPUTS: node, servers, $C_{s\rightarrow proc}$, $C_{s\rightarrow mem}$, U_{nom}
2. Evaluate P_{node} and M_{node} .
3. If the cloud has P_{node} and M_{node} resources to allocate the node then:
4. For each server in the input servers:
5. If the server has P_{node} and M_{node} resources to allocate the node:
6. Select this server, find the power consumed on the selected server for the node and exit.
7. OUTPUTS: the selected server and its power on success and null on failure.

Figure 16 – Algorithm for finding an assignable server in a cloud

Figure 17 encapsulates the entire partitioning stage. It incorporates the four main components, which are discussed above. The deployment from the initial deployment stage is taken as the starting point. Then, R_{net_delay} is evaluated for this deployment. A user of the MCAD algorithm can specify the window size (P_{window}) which determines the number of nodes to evaluate before making a move. P_{window} is analogous to a sliding or moving window. Instead of considering all the nodes before making a move, the sliding window provides control on the number of nodes to consider before making a

move. For each node in the moving window, the gain and power are calculated for a potential move to every other cloud. Among all the potential moves, only two moves are recorded. The move with the highest gain is recorded, which is named as the best gain move. This move reduces R_{net_delay}. Also, the move that results in the maximum power reduction is recorded only if the response time constraint is satisfied i.e. the sum of R_{PT} (see Equation 5.9) and R_{net_delay} is less than or equal to $R_{constraint}$. This move is named as the best power move. After evaluating every node in the sliding window, a move is taken. Priority is given to the best power move, if available. The best gain move is accepted if the best power move does not exist. This move selection strategy assures that a power efficient deployment, which satisfies the response time constraint, will be achieved at the end of the partitioning stage. The evaluation of nodes is repeated until the stop condition is met.

1. INPUTS: directed graph, deployment, clouds, user, root node, P_{window} , P_{max_moves} , R_{PT}, $R_{constraint}$
2. Moves taken ← 0.
3. Attempted tries ← 0.
4. Calculate R_{net_delay} for the deployment.
5. Sort the nodes in the directed graph in decreasing order of processing requirement.
6. Repeat:
7. Best power move {cloud, node, server, power, gain} ← null.
8. Best gain move {cloud, node, server, power, gain} ← null.
9. Find a moving window, i.e. a subset of nodes, using P_{window} .
10. For each node in the moving window:
11. Find the current_cloud where the node is assigned.
12. For each cloud in clouds:
13. Calculate the gain for moving the node from the current_cloud to cloud.
14. Find {server, power} for allocating the node in the cloud.
15. Record the best power move that causes maximum power reduction with respect to current allocation and does not violate the response time constraint.
16. Record the best gain move with the largest gain.
17. Increment attempted tries by 1.
18. If the best power move exists, then:
19. Increment moves taken by 1 and subtract the gain from R_{net_delay} .
20. Accept the best power move.
21. Else if the response time constraint is not met and best gain move exists then:
22. Increment moves taken by 1 and subtract the gain from R_{net_delay} .
23. Accept the best gain move.
24. Else:
25. No move is found.
26. If the stop condition is true, then exit.
27. OUTPUTS: the deployment if the response time constraint is met or null on failure.

Figure 17 – Algorithm for the partitioning stage

6.7 Stage 7: Uncoarsening

Unmerge any merged nodes to restore the original graph before coarsening (see the last

paragraph in Section 6.4).

6.8 Stage 8: Bin Packing: Before and After Uncoarsening

This section contains the details for the eighth stage of the algorithm where different bin packing strategies are utilized to further reduce the power consumed by the deployment found in the partitioning stage. The bin packing strategies reduce the number of servers used, and this should reduce the power consumed by the deployment. The following three bin packing strategies are used: BP-RUAEE, BP-HBF-Proc and BP-HBF-Mem. Section 2.7 has a general description of these strategies, and Section 5.1 provides an overview of these strategies in the context of this thesis.

Figure 18 describes the server selection process used in the three bin packing strategies. For the BP-RUAEE strategy, the bin packing score in line 10 of Figure 18 is modelled using Equations 1 and 2 in [13]. The server with the highest score is selected, and it has the most balanced resource utilization, i.e. the server's processing and memory capacities are uniformly utilized, if the node (2-D item) is assigned to it. For the BP-HBF-Proc and BP-HBF-Mem strategies, the bin packing score is evaluated using Equation 6.2, which shows the processing and memory capacities utilized in a server. The server with the highest score is selected, and it has the highest resource utilization if the node is assigned to it. In other words, the unused resources on a server are the smallest once the node is assigned to it. For a node, the processing (P_{node}) and memory (M_{node}) requirements are evaluated using Equations 5.10 and 5.11 respectively. Refer to Section 5.3.5 for details about $C_{s\rightarrow proc}$, $C_{s\rightarrow mem}$ and U_{nom}.

$$Bin\ Packing\ Score = \frac{CPU\ Utilization + Memory\ Utilization}{2} \tag{6.2}$$

1. INPUTS: node, servers, $C_{s\rightarrow proc}$, $C_{s\rightarrow mem}$, U_{nom}
2. Best server ← null.
3. Evaluate P_{node} and M_{node} .
4. If the cloud has sufficient processing and memory resources to allocate the input node then:
5. Best score ← -1.
6. For each server in the input servers:
7. If the server has sufficient processing and memory resources to allocate the node then:
8. CPU utilization ← (server's used $ssj\ ops$ + P_{node}) / server's total $ssj\ ops$.
9. Memory utilization ← (server's used memory + M_{node}) / server's total memory.
10. Calculate the bin packing score using the CPU utilization and memory utilization.
11. If (score > best score) then:
12. Best score ← score.
13. Best server ← server.
14. If the best server is not set, then fail since no server can allocate the node.
15. Evaluate the power consumed if the node is assigned to the best server.
16. OUTPUTS: the best server and power consumed on success, and null on failure.

Figure 18 – Algorithm to find a server in a cloud for the bin packing strategies

Figure 19 describes the generic form of the bin packing algorithm. It uses a deployment, which satisfies the response time constraint, as the starting point. The nodes are reassigned to the servers within a cloud using a bin packing strategy. The bin packing strategy can be either BP-RUAEE, BP-HBF-Proc or BP-HBF-Mem. In Figure 19, line 9 is the placeholder for the algorithm in Figure 18. For the BP-HBF-Mem strategy, line 6 in Figure 19 is amended; instead of sorting the nodes in decreasing order of the processing requirement, the nodes are sorted in decreasing order of the memory requirement.

The deployment can be either from the partitioning or uncoarsening stage. If the deployment from the partitioning stage is used, then the bin packing strategies are labelled as BP-RUAEE-Coarsen, BP-HBF-Proc-Coarsen and BP-HBF-Mem-Coarsen. If

the deployment from the uncoarsening stage is used, then the bin packing strategies are labelled as BP-RUAEE-Uncoarsen, BP-HBF-Proc-Uncoarsen and BP-HBF-Mem-Uncoarsen.

```
1. INPUTS: deployment, clouds, bin packing strategy
2. Total power ← 0.
3. Instantiate a new deployment to store the bin pack deployment.
4. For each cloud in clouds:
5.     Sort the nodes assigned to the cloud in decreasing order of processing requirement.
6.     Instantiate a replica of the cloud with no node allocations.
7.     For each node in the sorted set:
8.         Find {server, power} in the replica cloud to allocate the node using the selected bin
           packing strategy.
9.         If a server is not found, then fail and exit since no resources are available in the replica
           cloud to allocate the node.
10.        Add an entry {node, server, power} to the list associated with the cloud in the new
           deployment.
11.        Increment total power by power.
12. OUTPUTS: the new deployment and the total power on success, and null on failure.
```

Figure 19 – Algorithm for bin packing

Chapter 7 Tuning the MCAD Algorithm

The tuning objective is to find an effective and efficient configuration for the MCAD algorithm. The following three algorithm stages are explored to derive an effective and efficient configuration for the MCAD algorithm: the coarsening, partitioning and bin packing stages. The configuration for the MCAD algorithm depends upon the following choices:

1. Should coarsening be enabled or disabled?
2. What P_{window} value should be used during the partitioning stage?
3. Which bin packing strategies should be used: BP-RUAEE, BP-HBF-Proc or BP-HBF-Mem?
4. Should bin packing be performed before or after coarsening?

7.1 Applications: LQN Models

Five LQN models were chosen for tuning the algorithm. These models vary in width and depth, where the width is the number of LQN tasks at the same layer, and the depth is the number of layers. More variations in the width are introduced by varying the throughput requirement, which introduces more replicas of an LQN task. Also, a variety of patterns are included for the calls between the LQN tasks, which helps in covering a variety of communication workloads seen in the distributed applications. Figure 20 and Figure 21 show the first tuning model in the graphical and text forms respectively. Appendix B contains the remaining four tuning models:

- **Figure 27** – Tuning Model 2 (-A8)
- **Figure 28** – Tuning Model 3 (-A18)
- **Figure 29** – Tuning Model 4 (-A24)

- **Figure 30** – Tuning Model 5 (-A30)

The tuning LQN models were generated using a tool named *lqngen*, which is part of the LQNS software package [23]. *lqngen* generates random models. The *-A[ARG], --automatic=[ARG]* option, is used to generate the above-mentioned models. *ARG* determines the number of layers, clients, processors and tasks in the LQN model.

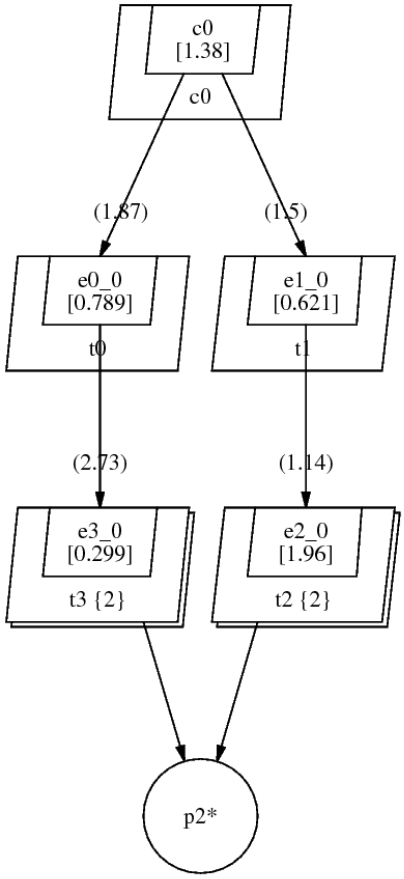

Figure 20 – Tuning Model 1 (-A4)

```
G "lqngen --automatic=4 --models=5 --request-rate=2" 1e-05 50 10 0.9 -1

P 4
  p c0 f m 1
  p p0 s 0.1 m 3
  p p1 s 0.1
  p p2 s 0.1
-1

T 5
  t c0 r c0 -1 c0
  t t0 n e0_0 -1 p0
  t t1 n e1_0 -1 p1
  t t2 n e2_0 -1 p2 m 2
  t t3 n e3_0 -1 p2 m 2
-1

E 5
# ---------- c0 ----------
  s c0   1.37556  -1
  y c0   e0_0 1.86668  -1
  y c0   e1_0 1.49936  -1
# ---------- t0 ----------
  s e0_0 0.788616 -1
  y e0_0 e3_0 2.72926  -1
# ---------- t1 ----------
  s e1_0 0.621402 -1
  y e1_0 e2_0 1.13872  -1
# ---------- t2 ----------
  s e2_0 1.95964  -1
# ---------- t3 ----------
  s e3_0 0.299109 -1
-1
```

Figure 21 – Text form of the LQN model shown in Figure 20

7.2 Bounds on the Solution Quality

A bound can either be an upper bound or a lower bound for a set. Bounds are values which can be easily evaluated without excessive computation. Also, there is no existing solution, which solves the *k* cloud deployment problem discussed in this thesis, to compare with the MCAD algorithm. So, the use of bounds is an easy and effective alternative to evaluate the quality of the MCAD algorithm's solution. The two important

characteristics of a deployment are the power consumption and response time per user request. For these two characteristics, the lower bound should be used as the baseline for evaluating the MCAD algorithm's solution.

7.2.1 Power Consumption Lower Bound

The lower bound on an application's power consumption is evaluated by allocating resources for the application on the most power efficient servers, which are operated at the optimal utilization where the ratio of the throughput and power is the highest. Furthermore, a fluid model is used to allocate the resources in order to avoid resource fragmentation. In the fluid model, the distributed application is considered as one single component with a processing and memory requirement, which fill the most power efficient servers and overflow to the next server until it is all allocated.

1. INPUTS: total application *ssj ops*, total application memory, servers from all the clouds
2. Power used ← 0.
3. Sort the servers in decreasing order of their highest throughput to power ratio.
4. For each server in servers:
5. Evaluate the utilization for the server's highest throughput and power ratio.
6. Allocate *ssj ops* and memory on the server based upon the evaluated utilization.
7. Increment power used based upon the allocated *ssj ops* on the server.
8. Decrement the total application *ssj ops* by the amount of allocated *ssj ops*.
9. Decrement the total application memory by the amount of allocated memory.
10. Exit if the total application *ssj ops* and memory are zero.
11. OUTPUTS: the power used on success and negative value on failure.

Figure 22 – Algorithm for evaluating the lower bound for an application's power consumption

In the fluid model, the concept of application components is ignored. For example, 40% of the resources for an application component can be allocated on one server, and the remaining 60% of the resources can be allocated on another server. The

network delay in the distributed application is ignored. The location of a server on a specific cloud is also ignored. This allows fractional resources to be allocated for an application component on two different servers, which can be located on two different clouds. Figure 22 shows the algorithm for evaluating the lower bound for an application's power consumption. The inputs include the total processing and memory requirements for the application, and the servers from all the clouds. The output is the theoretical lower bound for the power consumption, P_{bound}.

7.2.2 Response Time Per User Request Lower Bound

A lower bound on the response time per user request could not be found for an application. Two approaches were considered for finding a lower bound on the response time per user request. The first approach is similar to the initial deployment strategy where the directed graph nodes are assigned to the cloud closest to the current cloud if no resources are available on the current cloud. The first cloud chosen is the closest to the user. The closest cloud refers to the cloud with the least network delay. The second approach is similar to the partitioning stage, but the goal is to only minimize R_{net_delay} for the deployment. It was found that both approaches did not guarantee a lower bound on the response time per user request. Furthermore, the second approach is too complicated for a bound calculation. Ultimately, no simple approach was found, which guaranteed a lower bound on the response time per user request for an application.

7.3 Experimental Setup

This section describes the experiments used for tuning. It also outlines the criteria for evaluating the quality of a solution. In addition, it describes the hardware and software details involved in the setup, implementation and testing of the algorithm. At the

end, it includes a summary of the data collected from the tuning experiments.

7.3.1 Tuning Experiments

Three experiments are performed for each of the five LQN models mentioned in Section 7.1:

- In **experiment 1**, the throughput requirement (τ) is varied with easily achievable response time constraint.
- In **experiment 2**, the response time constraint ($R_{constraint}$) is varied from an easy target to a hard target.
- In **experiment 3**, the window size of the partitioning stage (P_{window}) is varied from one directed graph node to all nodes.

The goal of the three experiments is to study the impact of τ, $R_{constraint}$ and P_{window} on R_{MCAD}, P_{MCAD} and T_{MCAD} (defined in Section 5.2). Also, the three experiments are performed with and without coarsening in order to study the impact of coarsening on R_{MCAD}, P_{MCAD} and T_{MCAD}. The percentage difference is used for comparative analysis. Equation 7.1 evaluates the percentage difference between the values of an attribute of a solution under two different configurations, A and B.

$$\Delta_{\%}(A,B) = \left(\frac{A-B}{A}\right) \times 100\% \tag{7.1}$$

For evaluating the quality of a solution, the following criteria are used:

1. Is the response time constraint satisfied, ($R_{MCAD} \leq R_{constraint}$)?
2. How close is the solution deployment's power consumption, P_{MCAD} to the theoretical power lower bound, P_{bound}? The closeness can be measured using $\Delta_{\%}(P_{best}, P_{bound})$.
3. What is the MCAD algorithm's runtime, T_{MCAD}?

4. Has the deployment satisfied the application's resource requirements?

7.3.2 Hardware Details

The experiments are run on a machine with the following characteristics:

- Processor: Intel i5 3570, 3.4 GHz.
- Memory (RAM):16 GB.

7.3.3 Software Details

- The algorithm is implemented in Java.
- The Java Universal Graph Framework (JUNG) [24] is used for graph data structures.
- JSON.simple [25] is used for JSON processing.
- Guava [26], Google Core Libraries for Java, provide the Multimap data structure. It is used to represent the deployment. Multimap maps a key to multiple values. In the implementation, the key is the cloud, and the values are the {node, server, power} entries associated to the cloud.
- The Apache Commons Mathematics Library [27] is used to generate the power versus throughput regression model for the servers.
- The Eclipse IDE [28] is used for development.
- OpenJDK 8 [29], powered with OpenJ9 Java Virtual machine, is used to run the Eclipse IDE and the algorithm.
- Ubuntu 16.04 [30], a Linux operating system, is used.

7.4 Control Parameter Settings

In the current implementation, the following control parameters either retain static values or are ignored:

- U_{nom} is set to 0.8.
- In Table 21, the servers have either 16 GB, 64 GB, 128 GB or 192 GB of memory capacity. For a merged node to fit on all servers, B_{mem} is chosen to be 16 GB. In Table 21, the servers have either 8 CPUs, 72 CPUs, 88 CPUs or 112 CPUs. For a merged node to fit on all servers, B_{cpu} is chosen to be the average throughput of 8 CPUs. Its unit is $ssj\ ops$.
- B_{edge_factor} is set to 1. Thus, B_{edge_weight} is the average edge weight of the scaled directed graph.
- B_{node_count} is ignored since B_{cpu} and B_{mem} are sufficient to control the granularity of the coarsened graph.
- P_{max_moves} is ignored so there is no restriction on the maximum moves per node during the partitioning stage.

The impact of varying the above control parameters is not studied in the tuning phase.

7.4.1 Data Collection

Appendix C contains data from the tuning experiments. There are ten tables for each tuning model. Table 1 contains the cross-references to the tables corresponding to each tuning model. In all the tables in Appendix C, the first column represents the index for the case scenario. The details for each type of table are provided below:

Table sets 1, 3 and 4 are used to study the impact of varying τ and coarsening on R_{MCAD}, P_{MCAD} and T_{MCAD}.

- **Table set 1** contains the setup information for **experiment 1**, where τ is varied. The setup information includes τ, P_{window}, P_{max_moves}, B_{edge_factor}, B_{node_count} and $R_{constraint}$.
- **Table set 2** contains the application characteristics, which are derived from the setup information in table set 1. The application characteristics include $N_{coarsen}^{nodes}$, $N_{no-coarsen}^{nodes}$, M_{total} and P_{total}.
- **Table set 3** contains the results for **experiment 1** with coarsening. The results include N_{bad}^{ID}, P_{MCAD}, P_{bound}, $\Delta_{\%}(P_{MCAD}, P_{bound})$, solution $ID_{...}$, solution $D_{...}$, R_{MCAD} and T_{MCAD}.
- **Table set 4** contains the results for **experiment 1** without coarsening. The columns in table set 4 are identical to the columns in table set 3.

Table sets 5, 6 and 7 are used to study the impact of varying $R_{constraint}$ and coarsening on R_{MCAD}, P_{MCAD} and T_{MCAD}.

- **Table set 5** contains the setup information for **experiment 2**, where $R_{constraint}$ is varied. The columns in table set 5 are identical to the columns in table set 1.
- **Table set 6** contains the results for **experiment 2** with coarsening. The columns in table set 6 are identical to the columns in table Set 3.
- **Table set 7** contains the results for **experiment 2** without coarsening. The columns in table set 7 are identical to the columns in table set 3.

Table sets 8, 9 and 10 are used to study the impact of varying P_{window} and coarsening on R_{MCAD}, P_{MCAD} and T_{MCAD}.

- **Table set 8** contains the setup information for **experiment 3**, where P_{window} is varied. The columns in table set 8 are identical to the columns in table set 1.

- **Table set 9** contains the results for **experiment 3** with coarsening. The columns in table set 9 are identical to the columns in table set 3.
- **Table set 10** contains the results for **experiment 3** without coarsening. The columns in table set 10 are identical to the columns in table set 3.

Table 1 – Cross-references to the tables in Appendix C

	Tuning model 1	Tuning model 2	Tuning model 3	Tuning model 4	Tuning model 5
Table set 1	Table 22	Table 32	Table 42	Table 52	Table 62
Table set 2	Table 23	Table 33	Table 43	Table 53	Table 63
Table set 3	Table 24	Table 34	Table 44	Table 54	Table 64
Table set 4	Table 25	Table 35	Table 45	Table 55	Table 65
Table set 5	Table 26	Table 36	Table 46	Table 56	Table 66
Table set 6	Table 27	Table 37	Table 47	Table 57	Table 67
Table set 7	Table 28	Table 38	Table 48	Table 58	Table 68
Table set 8	Table 29	Table 39	Table 49	Table 59	Table 69
Table set 9	Table 30	Table 40	Table 50	Table 60	Table 70
Table set 10	Table 31	Table 41	Table 51	Table 61	Table 71

7.5 Observations from the Tuning Experiments

In this section, the data in Appendix C are evaluated in order to derive the tuned MCAD algorithm.

7.5.1 Experiment 1: Vary the Throughput Requirement

In experiment 1, τ is varied for the five tuning models. It is observed that P_{MCAD} increases as τ is increased. This is expected since more replicas of the application components are introduced as τ is increased. More servers are needed to support the application. The usage of more servers leads to the increase in the power consumption.

Overall, $\Delta_{\%}(P_{MCAD}, P_{bound})$ ranges between 3.3% and 26.4%. The mean and median for $\Delta_{\%}(P_{MCAD}, P_{bound})$ is 9.71% and 5.74% respectively. These are good results

considering that P_{bound} is evaluated in an extremely optimistic manner where the segregation between the application components and resource fragmentation are ignored.

For those throughputs for which the application required greater than ~25% of the total cloud resources, $\Delta_{\%}(P_{MCAD}, P_{bound})$ is less than ~10%. For those throughputs for which the application required less than ~25% of the total cloud resources, $\Delta_{\%}(P_{MCAD}, P_{bound})$ is between 10% and 26%. This behavior happens because the applications with large resource requirements better utilize the servers, and the servers tend to operate near the optimal power efficient state.

$R_{constraint}$ is always satisfied by the solution deployment. In most cases, all 52 initial deployments converged to a solution satisfying $R_{constraint}$. In case of tuning model 2, two initial deployments failed to satisfy $R_{constraint}$ at the highest τ setting, which corresponds to case 7 in Table 34 and Table 35. The two initial deployments that failed to satisfy the response time constraint are ID_{power} and ID_{delay}. This is an example where the initial deployments are incompatible with an application.

It is observed that T_{MCAD} increases as τ is increased. Increasing τ increases the graph size for the application. More replicas introduce more nodes and edges in the application's graph. It is informally observed that T_{MCAD} is proportional to the size of the application's graph i.e. T_{MCAD} increases roughly linearly with the graph size. This agrees with the $O(m)$ time complexity of the FM heuristic.

Experiment 1 is performed with and without coarsening. It is seen that coarsening helps achieve ~25% better R_{MCAD} values at an average in comparison to the no coarsening counterparts. Also, T_{MCAD} reduces by ~20% to 58% with coarsening.

P_{MCAD} achieved, with and without coarsening, is comparable. In most cases, the

difference in P_{MCAD} between the coarsening and no coarsening counterparts is $\pm 10W$. $10W$ is considered small since P_{MCAD} ranges between $80W$ to $1600W$ in Appendix C. There are a few outliers where P_{MCAD} is 5% to 10% better with coarsening. But there are similar outliers where P_{MCAD} is 5% to 10% better without coarsening. If these outliers are excluded, it can be concluded that coarsening has no impact on P_{MCAD}.

Overall, coarsening should be used because it gives better response times (R_{MCAD}) and algorithm runtimes (T_{MCAD}), with about the same result for the power consumption (P_{MCAD}).

7.5.2 Experiment 2: Vary the Response Time Constraint

In experiment 2, a use-case from experiment 1 is taken. In Table 1, table set 5 contains the chosen use-case for each tuning model. For the chosen use-case, $R_{constraint}$ is tightened until none of the 52 initial deployments can satisfy $R_{constraint}$. Also, experiment 2 is performed with and without coarsening.

Consistent with the observations of experiment 1, it is noticed that coarsening helps satisfy a more stringent $R_{constraint}$. For example, the lowest $R_{constraint}$ satisfied for tuning model 1 with coarsening is 300 milliseconds whereas the lowest $R_{constraint}$ satisfied without coarsening is 600 milliseconds. This behavior is observed with all the tuning models.

Similarly, more initial deployments satisfy $R_{constraint}$ with coarsening. In case of tuning model 2 and $R_{constraint}$ of 5000 milliseconds, 14 initial deployments satisfied $R_{constraint}$ with coarsening and only 2 initial deployments satisfied $R_{constraint}$ without coarsening. This behavior is seen with all the tuning models. Coarsening increases the success rate of the MCAD algorithm.

Deployments with better P_{MCAD} values are achieved with coarsening as $R_{constraint}$ is tightened. For example, in case of tuning model 5 (case 6 in Table 67 and Table 68), coarsening produced a deployment with 4% lower P_{MCAD}. This suggests that coarsening also reduces resource fragmentation on the servers. In other words, there are fewer unused resources on the active servers. Fewer servers will be needed if each server is as full as possible. Ultimately, this reduces P_{MCAD}.

In experiment 2, T_{MCAD} cannot be used to compare the quality of the solutions. T_{MCAD} decreases as the number of initial deployments which fail to satisfy $R_{constraint}$ increases. The number of initial deployments that fail to satisfy $R_{constraint}$ varies between the cases. But a trend is observed while comparing the cases where the number of initial deployments which violate the response time constraint are low and comparable. The trend suggests that T_{MCAD} increases as $R_{constraint}$ is tightened.

As seen in experiment 1, experiment 2 also provides evidence in favor of enabling coarsening. Coarsening achieves a tighter $R_{constraint}$, increases the success rate of the algorithm, and achieves a better P_{MCAD} as $R_{constraint}$ is tightened.

7.5.3 Experiment 3: Vary P_{window} in the Partitioning Stage

P_{window} represents the size of the moving window in the partitioning stage. A larger P_{window} corresponds to a greater effort in the partitioning stage. In experiment 3, a use-case is taken from experiment 2. In Table 1, table set 8 contains the selected use-case for each tuning model. For the selected use-case, P_{window} is varied for the partitioning stage. It is observed that a small P_{window} value may not guarantee a deployment which satisfies $R_{constraint}$. This behavior is seen in the case of tuning model 2 (Table 40, cases 1 and 2). The algorithm does not yield a deployment for tuning model 2 when $P_{window} =$

1 and $P_{window} = 2$. On the contrary, the largest P_{window} value (full window size) always yields a deployment, which satisfies $R_{constraint}$, for the five tuning models. A full window size refers to the case where P_{window} is equivalent to the total number of nodes in the application's graph.

In most of the cases, varying P_{window} does not have an impact on P_{MCAD}. But there are a few outliers where a smaller P_{window} will yield a lower P_{MCAD} in comparison to the full window size. In case of tuning model 1 (Table 30, cases 1 and 6), a deployment with 3% less P_{MCAD} is achieved with $P_{window} = 1$ in comparison to the deployment achieved with the full window size. Overall, no consistent pattern is observed that suggests a smaller P_{window} will yield a deployment with lower P_{MCAD}.

It is observed that T_{MCAD} is lower for smaller P_{window} values, and it increases as P_{window} increases. This behavior is expected since a small P_{window} value corresponds to less work in the partitioning stage.

In summary, P_{window} has no impact on P_{MCAD}. Lowering P_{window} reduces T_{MCAD}. But a smaller P_{window} does not always guarantee a deployment solution. On the other hand, the full window size always yields a deployment solution. Since the goal is to make the MCAD algorithm robust in terms of producing a deployment solution, the full window size is used in the tuned MCAD algorithm.

7.5.4 Comparing the Bin Packing Strategies

In Appendix C, a total of 154 cases are recorded between the five tuning models and three experiments. In this section, the seven bin packing strategies are evaluated across the 154 cases. Table 2 contains a summary of cases where each bin packing strategy yielded a deployment with the least power consumption. Table 3 is an extension

of Table 2. It contains the percentage differences in power consumption with respect to the best bin packing strategy for a specific case.

How to read Table 3? The row corresponding to "Table 48 (Case 4)" is chosen as an example. In case 4 of Table 48, $D_{BP\text{-}RUAEE\text{-}Coarsen}$ has the least power consumption. So, $D_{BP\text{-}RUAEE\text{-}Coarsen}$'s power consumption is used as the baseline in evaluating the percentage differences. For case 4 in Table 48, $D_{partition}$'s power consumption is 2.96% greater than $D_{BP\text{-}RUAEE\text{-}Coarsen}$'s power consumption, $D_{BP\text{-}HBF\text{-}Proc\text{-}Coarsen}$'s power consumption is equal to $D_{BP\text{-}RUAEE\text{-}Coarsen}$'s power consumption, $D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$'s power consumption is 0.23% greater than $D_{BP\text{-}RUAEE\text{-}Coarsen}$'s power consumption, and the power consumption of $D_{*\text{-}Uncoarsen}$ variants is exactly equal to the power consumption of the $D_{*\text{-}Coarsen}$ variants. "Baseline" is used for the deployment with the least power consumption, and "Failed" is used if a deployment is not found.

The following observations are made from Table 2 and Table 3:

- In Table 2, it is observed that the partitioning stage, which employs the FFDH bin packing strategy, provides a deployment with the least power consumption in 85% of the cases. The BP-HBF-Mem strategy yields a deployment with the least power consumption in 10% of the cases. The BP-HBF-Proc and BP-RUAEE strategies yield a deployment with the least power consumption in the remaining 5% of the cases.
- In Table 3, there is negligible difference in the power consumption between the $D_{*\text{-}Coarsen}$ and $D_{*\text{-}Uncoarsen}$ variants. Also, this phenomenon is noticed in all the cases in Appendix C. So, it makes no difference to run a bin packing strategy before or after

the uncoarsening stage.

- In Table 3, it is noticed that the BP-RUAEE and BP-HBF-Proc strategies always yield the same power. Also, this characteristic is seen in all the cases in Appendix C. Thus, the algorithm only needs to use either the BP-RUAEE or BP-HBF-Proc bin packing strategy.

Table 2 – Summary of best power with respect to the algorithm stages that produce a deployment

Deployment	Cases with best power	Instances
$D_{BP\text{-}RUAEE\text{-}Coarsen}$	4	Table 48 (Case 4); Table 51 (Case 4, Case 5); Table 71 (Case 1)
$D_{BP\text{-}HBF\text{-}Proc\text{-}Coarsen}$	1	Table 50 (Case 5)
$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	8	Table 37 (Case 4); Table 38 (Case 4); Table 40 (Case 3, Case 5); Table 61 (Case 2, Case 6); Table 70 (Case 6); Table 71 (Case 4)
$D_{BP\text{-}RUAEE\text{-}Uncoarsen}$	1	Table 70 (Case 1)
$D_{BP\text{-}HBF\text{-}Proc\text{-}Uncoarsen}$	0	None
$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	8	Table 40 (Case 6); Table 41 (Case 6); Table 50 (Case 3, Case 4); Table 51 (Case 1); Table 61 (Case 1); Table 71 (Case 2, Case 5)
$D_{partition}$	132	Refer to Appendix C.

- In Table 3, it is observed that the BP-RUAEE and BP-HBF-Proc strategies fail to find a deployment in some cases. The bin packing strategies are heuristic based,

so they do not guarantee the best solution all the time. These failing cases are outliers where the BP-RUAEE and BP-HBF-Proc strategies do not yield a good solution. The failing cases are seen when an application uses almost all the resources in a cloud and the partitioning stage tightly packs the application components onto a cloud with minimal resource fragmentation. If the BP-RUAEE and BP-HBF-Proc strategies underperform in these conditions, then they fail to find a deployment where the resource requirements of the application components are satisfied.

Table 3 – Percentage difference in the power consumption with respect to the MCAD algorithm stages that produce a deployment

Case	$D_{partition}$	$D_{BP\text{-}RUAEE\text{-}Coarsen}$	$D_{BP\text{-}HBF\text{-}Proc\text{-}Coarsen}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	$D_{BP\text{-}RUAEE\text{-}Uncoarsen}$	$D_{BP\text{-}HBF\text{-}Proc\text{-}Uncoarsen}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$
Cases where $\boldsymbol{D_{BP\text{-}RUAEE\text{-}Coarsen}}$ **(Baseline)** yielded a deployment with the least power.							
Table 48 (Case 4)	2.96	Baseline	0.00	0.23	0.00	0.00	0.23
Table 51 (Case 4)	2.51	Baseline	0.01	0.06	0.00	0.01	0.06
Table 51 (Case 5)	2.81	Baseline	0.00	0.13	0.00	0.00	0.13
Table 71 (Case 1)	9.86	Baseline	0.00	0.40	0.00	0.00	0.40
Cases where $\boldsymbol{D_{BP\text{-}HBF\text{-}Proc\text{-}Coarsen}}$ **(Baseline)** yielded a deployment with the least power.							
Table 50 (Case 5)	2.91	0.00	Baseline	0.11	0.26	0.26	0.08
Cases where $\boldsymbol{D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}}$ **(Baseline)** yielded a deployment with the least power.							
Table 37 (Case 4)	1.90	1.20	1.20	Baseline	1.27	1.27	0.18
Table 38 (Case 4)	1.75	Failed	Failed	Baseline	Failed	Failed	0.00
Table 40 (Case 3)	0.28	0.26	0.26	Baseline	0.31	0.31	0.31
Table 40 (Case 5)	2.44	Failed	Failed	Baseline	Failed	Failed	0.13
Table 61 (Case 2)	3.89	0.39	0.39	Baseline	0.23	0.23	0.02
Table 61 (Case 6)	3.29	0.01	0.01	Baseline	0.01	0.01	0.00

Case	$D_{partition}$	$D_{BP\text{-}RUAEE\text{-}Coarsen}$	$D_{BP\text{-}HBF\text{-}Proc\text{-}Coarsen}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	$D_{BP\text{-}RUAEE\text{-}Uncoarsen}$	$D_{BP\text{-}HBF\text{-}Proc\text{-}Uncoarsen}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$
Table 70 (Case 6)	3.04	0.91	0.91	Baseline	0.91	0.91	0.00
Table 71 (Case 4)	6.24	0.26	0.26	Baseline	0.26	0.26	0.00
Cases where $\boldsymbol{D_{BP\text{-}RUAEE\text{-}Uncoarsen}}$ **(Baseline)** yielded a deployment with the least power.							
Table 70 (Case 1)	3.93	1.80	1.80	0.11	Baseline	0.00	0.43
Cases where $\boldsymbol{D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}}$ **(Baseline)** yielded a deployment with the least power.							
Table 40 (Case 6)	2.31	Failed	Failed	0.06	Failed	Failed	Baseline
Table 41 (Case 6)	5.53	0.08	0.08	0.06	0.20	0.20	Baseline
Table 50 (Case 3)	3.52	1.57	1.57	1.45	1.59	1.59	Baseline
Table 50 (Case 4)	1.10	0.21	0.21	0.00	0.21	0.21	Baseline
Table 51 (Case 1)	0.06	0.08	0.08	0.00	0.08	0.08	Baseline
Table 61 (Case 1)	1.83	1.23	1.23	0.00	1.23	1.23	Baseline
Table 71 (Case 2)	8.24	0.44	0.44	0.00	0.44	0.44	Baseline
Table 71 (Case 5)	8.49	0.63	0.63	0.00	0.63	0.63	Baseline

7.6 The Tuned MCAD Algorithm

Based upon the observations recorded in Section 7.5, the following configuration is used for the tuned MCAD algorithm:

- Coarsening is enabled since it helps in satisfying tighter response time constraints, achieving lower response times, delivering deployments with lower power consumption, and reducing the MCAD algorithm's runtime.
- In the 154 cases across the three experiments, ID_{power} produced a solution deployment nine times, ID_{delay} produced a solution deployment one time, and $ID\text{-}_{randomX}$ produced a solution deployment in the remaining successful cases. All the three initial deployment strategies are used in the tuned algorithm since they produced a solution deployment at least once.
- The full window size is used to make the MCAD algorithm more robust since a smaller window size does not guarantee a deployment that will satisfy the response time constraint.
- There is a negligible difference in the power consumption between the $D_{*\text{-}Coarsen}$ and $D_{*\text{-}Uncoarsen}$ variants. So, the bin packing strategies are invoked once after the uncoarsening stage in the tuned algorithm.
- In 85% of the cases, $D_{partition}$ has the least power consumption. In 10% of the cases, $D_{BP\text{-}HBF\text{-}Mem\text{-}*}$ has the lowest power consumption. In the remaining 5% of the cases, $D_{BP\text{-}RUAEE\text{-}*}$ and $D_{BP\text{-}HBF\text{-}Proc\text{-}*}$ have the least power consumption. Also, it is noted that $D_{BP\text{-}RUAEE\text{-}*}$ and $D_{BP\text{-}HBF\text{-}Proc\text{-}*}$ always have the same power consumption. So, it is inefficient to use both the BP-RUAEE and BP-HBF-Proc strategies. Hence, only the BP-HBF-Proc and BP-HBF-Mem strategies are

employed in the tuned algorithm.

Figure 23 provides an overview of the tuned MCAD algorithm. The tuned MCAD algorithm produces 156 deployment candidates for the final solution due to the above changes. The unoptimized MCAD algorithm produces 364 deployment candidates for the final solution (Page 31). So, the tuned MCAD algorithm reduces the number of deployment candidates for the final solution by 208. Ultimately, this reduces T_{MCAD}.

1. INPUTS: clouds, servers, users, delays, application, control parameters
2. Select the clouds to deploy the application.
3. Generate a graph for the application.
4. Scale the application and generate a graph for the scaled application.
5. Coarsen the scaled application and generate a graph of the coarsened application.
6. For $i \leftarrow 1$ to number of initial deployments:
7. If $i = 1$, then generate a power-sensitive initial deployment.
8. Else if $i = 2$, then generate a delay-sensitive initial deployment.
9. Else generate a random initial deployment.
10. Partition the coarsened application graph using the initial deployment and the full window size.
11. If the partitioned deployment does not satisfy the response time constraint, then record the deployment and go to the next iteration in the loop.
12. Perform uncoarsening on the partitioned deployment and record the deployment.
13. For each {BP-HBF-Proc, BP-HBF-Mem}, perform bin packing on the uncoarsened partitioned deployment and record the deployment.
14. OUTPUTS: the deployment that consumes the least power and satisfies the response time constraint or null on failure.

Figure 23 – Overview of the tuned MCAD algorithm

Chapter 8 Validating the Tuned MCAD Algorithm

In this chapter, the objective is to validate the tuned algorithm using a larger set of LQN models. The criteria specified in Section 7.3.1 will be used to evaluate the quality of the MCAD algorithm's solution. In addition, a full LQN solution, in place of the approximation specified via Equations 5.9, 5.10 and 5.11, will be used to verify the response time of the solution deployment. The hardware and software details, specified in Sections 7.3.2 and 7.3.3 respectively, remain unchanged in this chapter.

8.1 LQN Models for Testing

In the previous chapter, the five tuning LQN models are generated using *lqngen*. For validating the tuned algorithm, 104 LQN models are generated using *lqngen*. The values for the *-A[ARG]*, *--automatic=[ARG]* option, which were used to generate the five tuning LQN models, are reused in generating the 104 LQN models. The *ARG* values for the *-A* option are 4, 8, 18, 24 and 30. In *lqngen*, the *-M[ARG]*, *--models=[ARG]* option is used to generate *ARG* different models. The *ARG* value for the *-M* option is set to 22. This generates 22 unique LQN models for each value of the *-A* option. With five different values for the *-A* option and *-M* option set to 22, *lqngen* generates 110 LQN models in total. The five tuning LQN models are removed from the 110 LQN models. Also, *lqngen* crashed while generating one LQN model. This gives 104 new LQN models, which are used to validate the tuned algorithm. Section D.1 contains the instructions to access any of the 104 models.

8.2 Response Time Verification by LQNS

The goal of this step is to verify the response time per user request of the MCAD algorithm's solution deployment (computed by the approximation of Equations 5.8 and

5.9) by comparison to the response time per user request estimated by a more accurate LQNS. Using the deployment solution, a new LQN model is generated with the details about the replicas, inter-cloud delays and server assignments. The new LQN model represents the solution deployment. The PS discipline is used in the new LQN model since it runs all the tasks simultaneously on the processor. Solving the new LQN model, using the LQNS, provides a value for the response time per user request. The response time per user request from the LQNS, R_{LQNS}, is compared with the response time per user request from the MCAD algorithm, R_{MCAD}. The percentage difference between R_{MCAD} and R_{LQNS}, $\Delta_{\%}(R_{MCAD}, R_{LQNS})$ (see Equation 7.1), is used for the comparative analysis in Section 8.5.4.

Why is this comparison performed? The MCAD algorithm approximates the wait times and resource contention by using a U_{nom}, nominal target utilization, of 0.8 for the servers. The usage of U_{nom} is seen in Equations 5.9 and 5.10. On the other hand, an LQNS evaluates the wait times and resource contention more accurately. So, an LQN solver's response time per user request is closer to the response time per user request from deploying an application in a real multi-cloud environment. The comparison of R_{MCAD} and R_{LQNS} will determine the quality of the above approximation for the wait times and resource contention.

Figure 24 and Figure 25 show the first half and second half of an LQN solution outline respectively. In Figure 24, N_{LQNS}^{iter} and T_{LQNS} are labelled. N_{LQNS}^{iter} represents the number of iterations taken by the LQNS to produce a solution; and T_{LQNS} represents the time taken by the LQNS to produce a solution. These two parameters quantify the effort taken by the LQNS to solve an LQN model. T_{LQNS} is presented in the

hours:minutes:seconds:ticks format, and it is converted into seconds in Table 74. A *tick* is equivalent to ten milliseconds. In Figure 25, R_{LQNS} is labelled, and its unit is milliseconds.

```
Convergence test value: 4.11239e-05
Number of iterations:   9  ← N_LQNS^iter

Pragmas:
    interlocking=none
    multiserver=rolia
    mva=schweitzer

 Solver: computer Linux 4.13.0-32-generic       Submodels:  5
     User:        0:06:11.64  ← T_LQNS          MVA Core():           45
     System:      0:00:00.00                    MVA Step():          285
     Elapsed:     0:06:11.64                    MVA Wait():  1.26361e+08
     MaxRSS:     7222392



Processor identifiers and scheduling algorithms:

Processor Name  Type    Copies  Scheduling Rate
c0              Inf       1     DELAY
delayP          Inf       1     DELAY
...

Task information:

Task Name       Type    Copies  Processor Name  Pri Entry List
c0              Ref(452)  1     c0                 c0
t2_26           Mult(452) 1     TX1330M2_3_4086    t2_26
...

Entry execution demands:

Task Name       Entry Name      Phase 1
c0              c0              0
t2_26           t2_26           1.07388
...

Mean number of rendezvous from entry to entry:

Task Name       Source Entry    Target Entry    Phase 1
c0              c0              delay_c0_t0_5   0.119041
                c0              delay_c0_t1_7   0.384687
...
```

Figure 24 – Sample LQNS Solution Outline [Part 1]

```
Type 1 throughput bounds:

Task Name       Entry Name      Throughput
c0              c0              0.228308
t2_26           t2_26           425.838
...

Mean delay for a rendezvous:

Task Name       Source Entry    Target Entry    Phase 1
c0              c0              delay_c0_t0_5   0
                c0              delay_c0_t1_7   4.24042e-05
...

Service times:

Task Name       Entry Name      Phase 1
c0              c0              1979.31  ← R_LQNS
t2_26           t2_26           1.04485
...

Service time variance (per phase)
and squared coefficient of variation (over all phases):

Task Name       Entry Name      Phase 1     coeff of var **2
c0              c0              3.19577e+06 0.815735
t2_26           t2_26           1.12665     1.032
...

Throughputs and utilizations per phase:

Task Name       Entry Name      Throughput  Phase 1     Total
c0              c0              0.228363    452         452
t2_26           t2_26           0.0282102   0.0294754   0.0294754
...

Utilization and waiting per phase for processor:  c0

Task Name       Pri n   Entry Name      Utilization Phase 1
c0              0   452 c0              0           0
...
```

Figure 25 – Sample LQNS Solution Outline [Part 2]

8.3 Validation Criteria

For validating the tuned algorithm, the following criteria specified in Section 7.3.1 are re-used:

1. Is the response time constraint satisfied, ($R_{MCAD} \leq R_{constraint}$)?
2. How close is the solution deployment's power consumption, P_{MCAD} to the

theoretical power lower bound, P_{bound}? The closeness can be measured using $\Delta_{\%}(P_{best}, P_{bound})$ (see Equation 7.1).

3. What is the MCAD algorithm's runtime, T_{MCAD}?
4. Has the deployment satisfied the application's resource requirements?

A new criterion is introduced for verifying the response time per request versus the LQNS result.

5. How does the response time per user request from the LQNS compare to the MCAD algorithm's response time per user request, $\Delta_{\%}(R_{MCAD}, R_{LQN})$?

8.4 Data Collection

Appendix D contains data for the tuned algorithm. There are three tables: Table 72, Table 73 and Table 74. Table 72 contains the setup information for the experiments, and the basic statistics for the application. Table 73 and Table 74 contain the results for the experiments that are specified in Table 72. In the three tables, the first column specifies an index corresponding to the case scenario. More details for each table are provided below:

- **Table 72:** The setup information includes the LQN model's name, τ, and $R_{constraint}$. The application characteristics include $N_{coarsen}^{nodes}$, M_{total}, Y_{total} and P_{total}.
- **Table 73:** The first part of the results includes N_{bad}^{ID}, P_{MCAD}, P_{bound}, $\Delta_{\%}(P_{MCAD}, P_{bound})$ solution $ID_{\dots}$, solution $D_{\dots}$, and T_{MCAD}.
- **Table 74:** The second part of the results includes R_{PT}, R_{delay}, R_{MCAD}, R_{LQN}, $\Delta_{\%}(R_{MCAD}, R_{LQN})$, T_{LQN} and N_{LQN}^{iter}.

8.5 Analysis

This section summarizes and evaluates the data in Appendix D. First, the 104 cases are profiled based upon the application traits such as $N_{coarsen}^{nodes}$, M_{total}, Y_{total}, P_{total} and R_{PT}. This captures the variation in the problem size. Then, the quality of the MCAD algorithm's solution deployment is evaluated by comparing P_{MCAD} with P_{bound}, $R_{constraint}$ with R_{MCAD}, and R_{MCAD} with R_{LQNS}. Finally, the algorithm's performance is evaluated using its runtime, T_{MCAD}.

8.5.1 Variety of Applications

The applications are characterized based upon traits such as the number of application components ($N_{coarsen}^{nodes}$), total processing requirement (P_{total}), total memory requirement (M_{total}), total number of calls per user request (Y_{total}) and the total processing time (R_{PT}). The above-mentioned traits need to be varied in order to thoroughly validate the tuned MCAD algorithm. This section verifies if the application traits are adequately varied across the 104 cases recorded in Appendix D.

Table 4 highlights the distribution of the number of application components ($N_{coarsen}^{nodes}$) across the 104 cases. In the first column, the number of components is split into bins. The second column shows the number of cases corresponding to a bin. The third column shows the percentage of cases corresponding to a bin. In Table 4, it is observed that the number of application components is varied sufficiently across the 104 cases. The smallest application has approximately ~20 components. The largest application has approximately ~240 components.

Table 4 – Statistics about the number of application components

Number of application components, $N_{coarsen}^{nodes}$	Number of cases	Percentage of cases [%]
20 - 25	3	2.88
25 - 50	9	8.65
50 - 75	22	21.15
75 - 100	30	28.85
100 - 125	18	17.31
125 - 150	12	11.54
150 - 175	6	5.77
175 - 200	2	1.92
200 - 225	1	0.96
225 - 240	1	0.96

Table 5 – Statistics about the total memory requirement of the applications

Total memory requirement, M_{total} [GB]	Number of cases	Percentage of cases [%]
30 - 60	3	2.88
60 - 120	11	10.58
120 - 180	15	14.42
180 - 240	31	29.81
240 - 300	23	22.12
300 - 360	11	10.58
360 - 420	7	6.73
420 - 480	2	1.92
480 - 520	1	0.96

Table 5 contains the distribution of the total memory requirement (M_{total}) across the 104 cases recorded in Appendix D. The first column contains the bins corresponding to the total memory requirement. Table 4 and Table 5 have identical second and third

columns. From Table 16, it is derived that the total memory capacity of all the clouds is 592 GB. In Table 5, it is noticed that the memory requirement is sufficiently varied across the 104 cases. The smallest memory requirement is ~30 GB, which is equivalent to 5% of the total memory capacity of all the clouds. The largest memory requirement is ~520 GB, which is equivalent to ~87% of the total memory capacity of all the clouds.

Table 6 contains the distribution of the total processing requirement (P_{total}) across the 104 cases recorded in Appendix D. The first column contains the bins corresponding to the processing requirement. Table 4 and Table 6 have identical second and third columns. From Table 16, it is derived that the total processing capacity of all the clouds is 15.40E+06 $ssj\ ops$. In Table 6, it is noticed that the processing requirement is adequately varied across the 104 cases. The smallest processing requirement is ~0.75E+06 $ssj\ ops$, which is equivalent to ~5% of the total processing capacity of all the clouds. The largest processing requirement is ~13.00E+06 $ssj\ ops$, which is equivalent to ~84% of the total processing capacity of all the clouds.

Table 7 contains the distribution of the total calls per user request (Y_{total}) in an application across the 104 cases recorded in Appendix D. The first column contains the bins corresponding to the total calls per user request. Table 4 and Table 7 have identical second and third columns. The total calls per user request represents the network delay in an application. In Table 7, it is observed that the total calls per user request is heavily varied across the 104 cases. The minimum total calls per user request is ~3. The maximum total calls per user request is ~11050. The large variation in total calls per user request helps to thoroughly evaluate the algorithm's partitioning stage.

Table 6 – Statistics about the total processing requirement of the applications

Total processing requirement, P_{total} $[ssj\ ops]$	Number of cases	Percentage of cases [%]
0.75E+06 – 2.00E+06	8	7.69
2.00E+06 - 4.00E+06	14	13.46
4.00E+06 - 6.00E+06	38	36.54
6.00E+06 - 8.00E+06	27	25.96
8.00E+06 - 10.00E+06	11	10.58
10.00E+06 - 12.00E+06	5	4.81
12.00E+06 – 13.00E+06	1	0.96

Table 8 contains the distribution of the total processing time (R_{PT}) of an application across the 104 cases recorded in Appendix D. The first column contains the bins corresponding to the total processing time of an application. Table 4 and Table 8 have identical second and third columns. In Table 8, it is observed that the total processing time is profoundly varied across the 104 cases. The minimum total processing time is ~8 milliseconds. The maximum total processing time is ~65 seconds. The large variation in execution delay covers a variety of applications. Real time applications, with quick response times, fall in the lower spectrum of the total processing time bins. Big data applications and grid computing applications, which require large computation times, fall in the higher spectrum of the processing time bins.

Table 7 – Statistics about the total calls per user request in the applications

Total calls per user request, Y_{total}	Number of cases	Percentage of cases [%]
3 - 25	19	18.27
25 - 50	4	3.85
50 - 75	10	9.62
75 - 100	10	9.62
100 - 150	6	5.77
150 - 200	5	4.81
200 - 250	4	3.85
250 - 300	4	3.85
300 - 350	2	1.92
350 - 400	3	2.88
450 - 500	1	0.96
550 - 600	4	3.85
650 - 700	2	1.92
700 - 1000	4	3.85
1000 - 2000	11	10.58
2000 - 3000	4	3.85
3000 - 4000	3	2.88
5000 - 5500	2	1.92
6000 - 6500	1	0.96
7000 - 7500	1	0.96
8000 - 8500	1	0.96
10000 - 11050	3	2.88

Table 4, Table 5, Table 6, Table 7 and Table 8 confirm that the five primary application traits, $N_{coarsen}^{nodes}$, M_{total}, Y_{total}, P_{total} and R_{PT}, are sufficiently varied across the 104 cases in Appendix D for validating the tuned MCAD algorithm.

Table 8 – Statistics about the total processing time of the applications

Total processing time, R_{PT} [seconds]	Number of cases	Percentage of cases [%]
0 - 0.1	19	18.27
0.1 - 1	35	33.65
1 - 5	24	23.08
5 - 10	11	10.58
10 - 20	7	6.73
20 - 30	2	1.92
30 - 40	3	2.88
40 - 50	1	0.96
50 - 60	1	0.96
60 - 70	1	0.96

8.5.2 P_{MCAD} versus P_{bound}

P_{bound} refers to the theoretical lower bound on the power consumption, and P_{MCAD} represents the lowest power consumption attained by the solution deployment of the MCAD algorithm. The percentage difference between P_{MCAD} and P_{bound}, $\Delta_{\%}(P_{MCAD}, P_{bound})$ (see Equation 7.1), reflects the quality of the solution produced by the MCAD algorithm in terms of the power consumption. Table 9 shows the distribution of $\Delta_{\%}(P_{MCAD}, P_{bound})$ across the 104 cases recorded in Appendix D. It is observed that ~77% of the cases have a $\Delta_{\%}(P_{MCAD}, P_{bound})$ below 10%, ~21% of the cases have a $\Delta_{\%}(P_{MCAD}, P_{bound})$ between 10% and 20%, and 2% of the cases have a $\Delta_{\%}(P_{MCAD}, P_{bound})$ between 25% and 27%.

When P_{MCAD} does poorly relative to P_{bound}, it is noticed that the application components do not utilize the servers near the optimal power efficient server state, which is the operating point corresponding to the server's highest throughput to power ratio.

This behavior is seen for applications with small resource requirements, which are insufficient to efficiently utilize the servers.

In some cases, tighter response time constraints can lead to partitioning moves, which primarily focus on reducing the total network delay (R_{net_delay}). This may prevent the application components to be deployed on the most power efficient machines across all the clouds. This is another reason for P_{MCAD} to perform poorly relative to P_{bound}.

Table 9 – Statistics about the percentage difference between P_{MCAD} and P_{bound}, $\Delta_{\%}(P_{MCAD}, P_{bound})$

Intervals of $\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Number of cases	Percentage of cases [%]
0 - 5	14	13.46
5 - 10	66	63.46
10 - 15	11	10.58
15 - 20	11	10.58
25 - 27	2	1.92

8.5.3 $R_{constraint}$ versus R_{MCAD}

$R_{constraint}$ refers to the response time constraint per user request, and R_{MCAD} refers to the response time per user request of the MCAD algorithm's solution deployment. Table 10 shows the distribution of the percentage difference between $R_{constraint}$ and R_{MCAD}, $\Delta_{\%}(R_{constraint}, R_{MCAD})$ (see Equation 7.1), across the 104 cases recorded in Appendix D. It is noticed that the MCAD algorithm always satisfies the response time constraint since $\Delta_{\%}(R_{constraint}, R_{MCAD})$ is always positive i.e. $R_{constraint} > R_{MCAD}$.

R_{MCAD} has two main components: R_{PT} and R_{net_delay}. R_{PT} is static, and it is not influenced in the MCAD algorithm. On the other hand, R_{net_delay} is reduced in the

partitioning stage. If R_{PT} does not violate $R_{constraint}$ by itself, then R_{net_delay} determines the quality of R_{MCAD} relative to $R_{constraint}$. R_{net_delay} depends on the application graph properties such as the number of edges and edge weights, which represent the communication between the application components. Y_{total} provides an overview of the edge properties since it represents the total communication in the application. For large values of Y_{total}, it is difficult to satisfy $R_{constraint}$ by reducing R_{net_delay}. In such cases, $\Delta_{\%}(R_{constraint}, R_{MCAD})$ is closer to 0%. For smaller values of Y_{total}, it is easier to reduce R_{net_delay}. In such cases, R_{MCAD} can be much lower than $R_{constraint}$.

Table 10 – Statistics about the percentage difference between $R_{constraint}$ and R_{MCAD}, $\Delta_{\%}(R_{constraint}, R_{MCAD})$

Intervals of $\Delta_{\%}(R_{constraint}, R_{MCAD})$ [%]	Number of cases	Percentage of cases [%]
0 to 1	34	32.69
1 to 5	18	17.31
5 to 10	6	5.77
10 to 25	13	12.50
25 to 40	8	7.69
40 to 55	9	8.65
55 to 70	8	7.69
70 to 85	4	3.85
85 to 95	4	3.85

The main objective of the algorithm is to satisfy the response time constraint per user request. The MCAD algorithm's ability to achieve a response time could be better tested if there was a simple approach to derive a lower bound on the response time per

user request. In Section 7.2.2, no simple approach was found that guaranteed a lower bound on the response time per user request. If such a lower bound could be easily calculated, then the MCAD algorithm could have been stress-tested in achieving a response time per user request at par to the lower bound on the response time per user request. One of the deficiencies of the validation phase is the lack of stress testing the response time per user request achieving capabilities of the MCAD algorithm.

8.5.4 R_{MCAD} versus R_{LQNS}

R_{LQNS} refers to the response time per user request from the LQNS (Section 8.2). R_{LQNS} accounts for the wait times and resource contention delay more accurately. This is comparable to the real life scenario of deploying the application. Ideally, R_{LQNS} should be equal to or less than R_{MCAD}. If R_{MCAD} is greater than R_{LQNS}, then the MCAD algorithm has overestimated the wait times and resource contention delay in R_{PT} (see Equation 5.9). If R_{MCAD} is less than R_{LQNS}, then the MCAD algorithm has underestimated the wait times and resource contention delay in R_{PT}.

Table 11 shows the distribution of the percentage difference between R_{MCAD} and R_{LQNS}, $\Delta_{\%}(R_{MCAD}, R_{LQNS})$ (see Equation 7.1), across the 104 cases recorded in Appendix D. $\Delta_{\%}(R_{MCAD}, R_{LQNS})$ is positive for the overestimated cases, and negative for the underestimated cases. In ~40% of the cases, the wait times and resource contention delay are overestimated. In ~39% of the cases, the wait times and resource contention delay are underestimated. In ~21% of the cases, the LQNS failed to solve the LQN model either due to a bad memory allocation or the internal limits that prevented the LQNS from handling large LQN models.

The application's resource requirements were compared with the clouds'

resources to justify the offsets between R_{MCAD} and R_{LQNS}. But no patterns were found that justified the offsets between R_{MCAD} and R_{LQNS}. For ensuring no server was overloaded, it was confirmed that the utilization of processors in the LQNS solution does not exceed U_{nom} (= 0.8). Thus, the offsets between R_{MCAD} and R_{LQNS} are regarded as approximation errors in R_{PT}.

Table 11 - Statistics about the percentage difference between R_{MCAD} and R_{LQN}, $\Delta_{\%}(R_{MCAD}, R_{LQNS})$

Intervals of $\Delta_{\%}(R_{MCAD}, R_{LQNS})$ [%]	Number of cases	Percentage of cases [%]
-47 to -40	1	0.96
-40 to -30	4	3.85
-30 to -20	6	5.77
-20 to -10	11	10.58
-10 to 0	18	17.31
0 to 10	31	29.81
10 to 20	8	7.69
20 to 30	2	1.92
40 to 45	1	0.96
LQNS Failures	22	21.15

Overestimating the wait times and resource contention delay is acceptable but underestimating them should be avoided in order to prevent a potential violation of the response time constraint. A safe buffer zone for $\Delta_{\%}(R_{MCAD}, R_{LQNS})$ can be a value greater than or equal to -10%. This allows a tolerance of 10% underestimation error in R_{MCAD}. In Table 11, 58% of the cases fall within the safe buffer zone, 21% of the cases fall outside the safe buffer zone, and the remaining 21% of the cases experience LQNS failures. This is a good distribution considering only 21% of the cases violate the safe buffer zone.

A simple approach to account for the approximation error in R_{PT} is to adjust $R_{constraint}$ using a multiplication factor of (R_{MCAD}/R_{LQNS}). A control parameter, α_{wait}, is introduced to specify this factor, which will increase $R_{constraint}$ and R_{net_delay} (see Equation 5.4) to handle an overestimation error in R_{PT}, and decrease $R_{constraint}$ and R_{net_delay} to handle an underestimation error in R_{PT}.

8.5.5 MCAD Algorithm Runtime, T_{MCAD}

T_{MCAD} is one of the crucial metrics to evaluate the performance of the MCAD algorithm. Table 12 highlights the distribution of the MCAD algorithm's runtime across the 104 cases recorded in Appendix D. In ~77% of the cases, the algorithm runtime is below 10 seconds. For ~12% of the cases, the algorithm runtime is between 10 seconds and 20 seconds. For ~11% of the cases, the algorithm runtime is between 20 seconds and 73.5 seconds.

Table 12 – Statistics about the MCAD algorithm's runtime, T_{MCAD}

MCAD algorithm's runtime, T_{MCAD} [seconds]	Number of cases	Percentage of cases [%]
0 - 1	16	15.38
1 - 5	53	50.96
5 - 10	12	11.54
10 - 20	12	11.54
20 - 30	4	3.85
30 - 40	2	1.92
40 - 50	2	1.92
50 – 73.5	3	2.88

The problem size depends upon the following five main factors: the number of clouds, total number of calls per user request in the application, number of application

components, and total processing and memory requirements of the application. In Figure 26, it is observed that T_{MCAD} increases roughly linearly with the number of application components. This agrees with the $O(m)$ time complexity of the FM heuristic because the partitioning stage represented a large share of T_{MCAD} in most cases. In few cases, scaling the application was expensive due to creating a graph with many nodes and edges, and performing recursive operations on the large graph. This is an implementation issue which can be improved to reduce the time taken during the scaling stage. But the scaling intensive cases create T_{MCAD} outliers, which disrupt the linear relationship between T_{MCAD} and problem size.

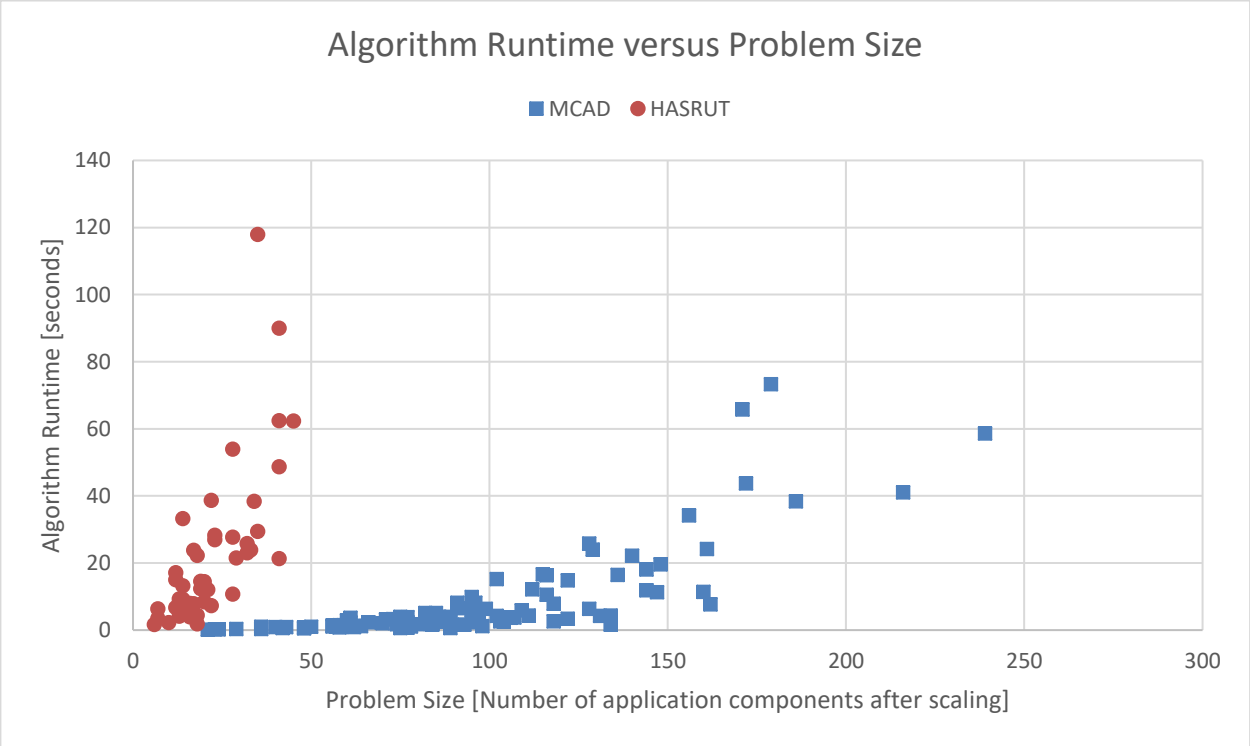

Figure 26 – Algorithm runtime versus problem size for the MCAD and HASRUT algorithms

The work on the HASRUT algorithm [7] quantifies the problem size by only using the number of application components. In comparison to the work on the HASRUT algorithm, this work covers a larger variety of problem sizes. In this work, the largest

problem size is ~5 times greater than the problem size explored by Kaur [7] for the HASRUT algorithm. Figure 26 shows the algorithm runtime versus problem size for the MCAD and HASRUT algorithms. For the HASRUT algorithm, the data is extracted from Table A.6 in [7]. In Section 6.5 of [7], it is noted that the HASRUT algorithm is tested on an Intel Core i7 3770 3.4 GHz processor with a 16 GB RAM. In Section 7.3.2, it is mentioned that the MCAD algorithm is tested on Intel Core i5 3540 3.4 GHz processor with a 16 GB RAM. Intel Core i5 3540 is less powerful in comparison to Intel Core i7 3770. So, the MCAD algorithm is tested on a slower processor. While considering the slower processor, Figure 26 illustrates that the MCAD algorithm outperforms the HASRUT algorithm in terms of the algorithm runtime.

Table 13 – Statistics about the LQNS's runtime, T_{LQNS}

LQNS's Runtime, T_{LQNS} [seconds]	Number of cases	Percentage of cases [%]
0 - 1	8	7.69
1 - 5	11	10.58
5 - 50	20	19.23
50 - 100	6	5.77
100 - 300	12	11.54
300 - 500	9	8.65
500 - 750	4	3.85
750 - 1000	2	1.92
1000 - 1250	2	1.92
1250 - 1500	4	3.85
1500 – 1730	4	3.85
LQNS Failures	22	21.15

The LQNS is an alternative to the MCAD algorithm for accurately evaluating the

response time per user request of an application. Table 13 shows the distribution of the LQNS's runtime, T_{LQNS}, across the 104 cases recorded in Appendix D. The LQNS's runtime only represents the evaluation of one deployment whereas the MCAD algorithm's runtime represents the evaluation of 156 deployments (Section 7.6). In Table 12, the maximum T_{MCAD} recorded is ~73.5 seconds. In Table 13, the maximum T_{LQNS} recorded is ~1730.0 seconds. Table 13 demonstrates the time-consuming nature of the LQNS in comparison to the MCAD algorithm.

Chapter 9 HASRUT versus MCAD

The MCAD algorithm is a significant extension to the HASRUT algorithm. To highlight the contributions of this work, it is vital to show the similarities and differences between the MCAD and HASRUT algorithms. Table 14 has three sections. The first section shows the similarities between the HASRUT and MCAD algorithms. The second section displays the improvements of the MCAD algorithm in comparison to the HASRUT algorithm. The last section describes the differences in the implementation and validation details between the HASRUT and MCAD algorithms.

Table 14 – Comparison between the HASRUT and MCAD algorithms

Criteria	HASRUT	MCAD
Section 1: Algorithm similarities		
Does it handle heterogeneous clouds i.e. clouds of different sizes?	Yes.	Yes.
Does it account for delays between clouds?	Yes.	Yes.
Does it account for delays within clouds?	No.	No.
Does the partitioning stage aim at satisfying the response time constraint?	Yes.	Yes.
Section 2: Algorithm differences		
Does it handle heterogenous servers i.e. servers with different processing and memory capacities?	No.	Yes.
How many clouds can it handle?	Two.	No limit (k clouds).
Can it deploy replicas of a task on different clouds?	No.	Yes.
What does the coarsening phase rely upon?	B_{edge_weight}.	B_{edge_weight}, B_{cpu}, B_{mem} and B_{node_count}.
Which initial deployment strategies does it support?	Supports only one strategy: the	Supports three strategies: the

Criteria	HASRUT	MCAD
	random flavor.	power, random and delay flavors, which are described in Section 6.5.
Does the partitioning stage aim at reducing power consumption?	No.	Yes.
Does it support multiple bin packing strategies?	Only supports the BP-HBF-RAM strategy.	Supports the BP-HBF-RAM, BP-HBF-PROC, BP-RUAEE and FFDH strategies.
Does it support bin packing before or after uncoarsening?	No.	Yes.
Section 3: Implementation and validation differences		
Which language is used to implement the algorithm?	Python.	Java.
Does the implementation utilize a graph library?	No.	Yes (JUNG) [24].
Can the implementation be easily extended to support other initial deployment, partitioning and bin packing strategies?	No. Procedural programming with code bloat is observed.	Yes. Object-oriented programming is used.
Can the implementation handle any LQN model?	No, it is constrained by certain requirements.	Yes, it can handle any random LQN model.
Is the lower bound on power consumption evaluated for quality inspection [Section 7.2.1]?	No.	Yes.
Does it support response time per user request verification using the LQNS [Section 8.2]?	No.	Yes.

Chapter 10 Conclusions and Future Work

10.1 Conclusions

The MCAD algorithm solves the multi-cloud deployment problem, finding deployments that consume little power while satisfying the response time and other constraints. In the 104 testcases on problems with four clouds and up to 240 deployable units (tasks), it found solutions with total power consumption within 10% of an idealized bound in 77% of the testcases, while respecting the response time constraint up to the point where it became infeasible with the total cloud resource capacity. 90% of the solution times took less than 20 seconds, and 80% took less than 10 seconds, even including the largest problems. Solution times increased with the problem size (deployable units, including replicas introduced to handle high system loads), roughly linearly (as shown in Figure 26). Figure 26 also shows that the MCAD algorithm is faster and scales better than the HASRUT algorithm, which can only deploy on two clouds.

A set of configuration options for the MCAD algorithm were investigated. It was shown that using coarsening/uncoarsening of the application graph considerably improves the quality of the solutions and reduces the time to obtain them. The new initial deployment strategies, the power-sensitive and delay-sensitive flavors, result in a solution in some of the test cases. One of the bin-packing strategies (BP-RUAEE-Coarsen) could safely be dropped as the other two (BP-HBF-Proc-Coarsen and BP-HBF-Mem-Coarsen) appeared to cover the cases equally well. The winning configuration scales the application by introducing replicas, coarsens the graph, creates a set of initial deployments to the clouds (including random deployments), adjusts the partitioning between clouds by considering all candidate nodes at every step, uncoarsens the graph

and applies two bin-packing strategies to obtain detailed deployments. This creates about 156 candidate deployment solutions, of which the best is chosen.

The response time constraint was imposed using a simple approximation during the deployment calculations, and then checked against the full LQN model for the solution deployment. The MCAD algorithm underestimated the response time with respect to the response time from the LQNS by more than 10% in 21% of the test cases, resulting in a recommendation to adjust the response time constraint by setting α_{wait} to an appropriate value before applying the MCAD algorithm, to compensate.

In this work, the application model is unable to support application components with varying service demands, which is a limitation. An example is replicated databases where the replicas need to stay synchronized. In such cases, the service demand varies with the synchronization workload, which will vary as the number of replicas of the database changes. In order to incorporate such workloads in the application model, they need to be studied and custom modelled.

10.2 Summary of Contributions

The main contribution is the MCAD algorithm, which is the only algorithm for *k*-way multi-cloud application deployment that reduces the power consumption while satisfying the SLA requirements on the throughput requirement and response time constraint. It would deliver the most value in lowering the costs for a cloud owner who is "application aware" and has the authority to redeploy, scale and migrate the application instances. Also, it targets "green computing" since there will be less environmental impact due to the reduction in the clouds' power consumption.

10.3 Future Work

10.3.1 Selecting the Clouds and Servers

The first step is to select the clouds and a subset of the servers in each cloud. In this work, a set of four clouds is chosen to portray the multi-cloud environment. In the real life, there may be many clouds available. So, the set of clouds to deploy an application can be chosen dynamically in real-time based upon the cloud's delay from the users, available resources and other properties. Also, the overall characteristics of a cloud, such as the power efficiency (PUE), availability and operational cost efficiency, can be used as a decision-making factor in selecting the clouds. The dynamic selection of clouds will help in automating the MCAD algorithm. Furthermore, it will improve the outcome of the MCAD algorithm by providing a more optimal multi-cloud environment for deploying an application.

Also, the initial selection of cloud resources (servers) can be dynamically performed in real-time based upon the application's resource requirement. This will allow the cloud resource providers to prioritize the customers based upon their SLA. The high priority users can be allowed more resources on the closer clouds. Also, dynamically selecting cloud resources will help in automating the MCAD algorithm. Furthermore, it can help in efficiently selecting a subset of the cloud resources, thereby, letting the remainder of the cloud resources be utilized for deploying other applications simultaneously.

10.3.2 Power Model

In this work, the power model considers the power consumed by the clouds' servers while assuming a constant PUE value across all the clouds. This allows the power

consumed by a cloud's servers to be a scaled down representation of the cloud's total power consumption. In a realistic setting, the PUE value will vary depending upon the cloud. The varying aspect of the PUE value can be incorporated in the power model in order to yield more realistic results.

10.3.3 Cloud Outages (Unreliability)

The clouds are vulnerable to outages due to "man-made causes (e.g., malicious attacks or administrator errors) and natural disasters (e.g., floods, tornados, etc.)" [1]. This work can handle cloud outages by reassigning the application deployable units (nodes) to other power efficient servers available within the affected cloud. If a cloud is completely shut down due to a long outage where even the power backups have failed, then the MCAD algorithm can be re-run by replacing the unavailable cloud with other available clouds. This will deploy the application to available servers on the unaffected clouds.

If an application is already deployed, then only the application deployable units, which are impacted by the outage, should be redeployed. This will keep redeployment costs low. This will require the server assignment of the unimpacted deployable units to stay unchanged in the MCAD algorithm.

Also, an initial deployment strategy can be introduced for assigning the application deployable units to servers based upon their probabilistic availability while giving preference to servers which are more likely to stay available. The same approach can be utilized during the cloud selection phase (stage 1) such that the clouds are selected based upon their probabilistic availability.

10.3.4 Multi User-Group Scenario

The MCAD algorithm only handles a single user-group, which is a set of users who have similar delays to the clouds and the same response time constraint. A multi user group scenario refers to multiple sets of users with different delays to clouds, response time constraints and throughput requirements. The MCAD algorithm does not account for multiple user groups. The following are two approaches to handle multiple user groups:

1. Multiple user groups can be handled by reusing the MCAD algorithm without any changes. For each user group, the application can be deployed separately and concurrently using the MCAD algorithm. Concurrency is possible if the cloud resources used for deploying each application instance are exclusive.

   Optimizations can be made in terms of merging the application resources after the deployments have been determined for each user-group. If security is a concern and each user group wants to avoid resource sharing, then optimizations related to resource merging can be avoided.
2. A more complicated approach is to modify the MCAD algorithm to handle multi user groups. The following stages of the MCAD algorithm will undergo changes to account for multiple user groups:
   a) During the application graph generation stage, a separate gateway will be introduced to handle requests from each user group.
   b) During the scaling stage, the aggregated throughput of all user groups will be used to determine the number of replicas.
   c) New initial deployment strategies, that are better suited for multiple user groups, will be needed.

d) During the partitioning phase, the gain, R_{net_delay} calculation and heuristics will change to account for multiple user-groups.

In my opinion, the second approach will not scale well, and the partitioning phase will find it more difficult to converge to a solution for multiple user groups.

10.3.5 Modelling I/O Delays and Disk Requirements

This work does not account for the disk I/O delays and disk storage requirements. Modelling these two application traits will improve the response time approximation in the MCAD algorithm. The disk I/O delays and disk storage requirements can be modelled explicitly using an LQN task. Each LQN task can be linked to another LQN task, which will represent the disk I/O delay and disk storage requirement. A constraint will be imposed to deploy an LQN task and its associated LQN task for the disk I/O delay and disk storage requirement together on the same computing device.

10.3.6 Network Delay Model

The MCAD algorithm considers constant delays between the entities, which can either be a cloud or user. This delay model does not account for the network load, congestion, and other properties, which may influence the delays between the entities. A more sophisticated delay model can be used to evaluate the delays between the entities using the real-time factors.

10.3.7 Deploying Multiple Applications Simultaneously

In Section 10.3.3, a strategy is mentioned to deploy an application for multiple user groups while reusing the MCAD algorithm. The same strategy can be used to deploy multiple applications separately and concurrently.

10.3.8 Interchanging Partitioning Metrics

In this work, the goal of the partitioning stage is to reduce the power consumption of the deployment while satisfying the throughput requirement and response time constraint. Currently, the MCAD algorithm supports green computing since the cloud infrastructure is responsibly utilized in terms of the power consumption. The cost of power varies based upon the geographic location and time of the day. If the cost of electricity is cheapest at a cloud with servers that inefficiently consume power, then more power can be consumed in a solution that aims to reduce the total cost of electricity utilized. The cloud resource providers may want to reduce the total cost of electricity utilized instead of reducing the total power consumption. This will require the partitioning stage to reduce for other metrics, which are dependent on the application graph's and cloud's properties, such as the total cost of electricity utilized or the total cost incurred by the application user. A feature to easily amend the partitioning stage heuristic will be useful in solving variants of the multi-cloud deployment problem.

10.3.9 Deploying in Real-life

In this work, the response time of the deployment solution produced by the MCAD algorithm is verified by the LQNS, which provides results comparable to real-life via simulation. The results of the MCAD algorithm are not verified by deploying an application in an actual multi-cloud environment. It was not possible to deploy an application in a real multi-cloud environment due to unavailability of multi-cloud resources and time constraints. But, the effectiveness of the MCAD algorithm can be improved by considering factors which have not been accounted for and are discovered by deploying an application in a real multi-cloud environment. Kubernetes [31], a

container orchestration tool, can be used to deploy an application in a real multi-cloud environment using the deployment solution generated by the MCAD algorithm. Two new components will be required to achieve this goal. The first component will be responsible for translating the MCAD algorithm's solution deployment into a Kubernetes deployment configuration. The second component will be responsible for creating Kubernetes clusters for each cloud. Each cloud cluster will comprise of the servers that are chosen to deploy the application. A software tool such as Rancher [32] can be used to easily deploy and manage Kubernetes clusters on any cloud provider. With these two new components, the solution deployment can be tested in a real multi-cloud environment.

References

[1] B. Lee, T. Grance, R. Patt-Corner and J. Voas, "Cloud Computing Synopsis and Recommendations," in *Special Publication 800-146*, National Institute of Standards and Technology (NIST), 2012.

[2] R. Chandramouli, "Security Strategies for Microservices-based Application Systems," in *Special Publication 800-204*, National Institute of Standards and Technology (NIST), 2019.

[3] Microsoft Azure, "What are public, private, and hybrid clouds?," 6 December 2018. [Online]. Available: https://azure.microsoft.com/en-ca/overview/what-are-private-public-hybrid-clouds/. [Accessed 7 December 2018].

[4] Nokia Networks, "Edge Cloud," 13 June 2018. [Online]. Available: https://networks.nokia.com/solutions/edge-cloud. [Accessed 3 December 2018].

[5] NIST Cloud Service Metrics Sub Group, "Cloud Computing Service Metrics Description," in *Special Publication 500-307*, National Institute of Standards and Technology (NIST), 2015.

[6] M. Rouse, "Real-time Application," WhatIs.com, April 2008. [Online]. Available: https://searchunifiedcommunications.techtarget.com/definition/real-time-application-RTA. [Accessed 2018 December 18].

[7] R. Kaur, "Lightweight Robust Optimizer for Distributed Application Deployment in Multi-Clouds," Master's Thesis, Carleton University, Ottawa, 2015.

[8] C. Möbius, W. Dargie and A. Schill, "Power Consumption Estimation Models for

Processors, Virtual Machines, and Servers," *IEEE Transactions on Parallel and Distributed Systems,* vol. 25, no. 6, pp. 1600-1614, 2014.

[9] Standard Performance Evaluation Corporation, "SPEC Power," 25 October 2018. [Online]. [Accessed 7 December 2018].

[10] B. Menegola, "A study of the k-way graph partitioning problem," Federal University of Rio Grande do Sul, Rio Grande do Sul, 2012.

[11] H. I. Christensen, A. Khan, S. Pokutta and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: A survey," *Computer Science Review,* vol. 24, pp. 63-79, 2017.

[12] British German Academic Research Collaboration Programme, "Survey on two-dimensional packing," September 2006. [Online]. Available: http://cgi.csc.liv.ac.uk/~epa/surveyhtml.html. [Accessed 5 December 2018].

[13] G. Han, W. Que, G. Jia and W. Zhang, "Resource-utilization-aware energy efficient server consolidation algorithm for green computing in IIOT," *Journal of Network and Computer Applications,* vol. 103, pp. 205-214, 2018.

[14] K. Molka and G. Casale, "Energy-efficient resource allocation and provisioning for in-memory database clusters," *IFIP/IEEE Symposium on Integrated Network and Service Management (IM),* pp. 19-27, 2017.

[15] C. Tunc, N. Kumbhare, A. Akoglu, S. Hariri, D. Machovec and H. J. Siegel, "Value of Service Based Task Scheduling for Cloud Computing Systems," *International Conference on Cloud and Autonomic Computing (ICCAC),* pp. 1-11, 2016.

[16] Y. Wang and Y. Xia, "Energy Optimal VM Placement in the Cloud," *IEEE 9th*

International Conference on Cloud Computing (CLOUD), pp. 84-91, 2016.

[17] M. E. Frincu and C. Craciun, "Multi-objective Meta-heuristics for Scheduling Applications with High Availability Requirements and Cost Constraints in Multi-Cloud Environments," *2011 Fourth IEEE International Conference on Utility and Cloud Computing, Victoria, NSW,* pp. 267-274, 2011.

[18] S. Panda and P. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *The Journal of Supercomputing,* vol. 71, no. 4, pp. 1505-1533, 2015.

[19] N. Verba, "Application Deployment Framework for large-scale Fog Computing Environments," PhD Thesis, Coventry University, Coventry, England, 2019.

[20] Standard Performance Evaluation Corporation, "SPECpower_ssj2008 Result File Fields," 7 July 2018. [Online]. Available: https://www.spec.org/power/docs/SPECpower_ssj2008-Result_File_Fields.html. [Accessed 6 December 2018].

[21] J. Yuventi and R. Mehdizadeh, "A critical analysis of Power Usage Effectiveness and its use in communicating data center energy consumption," *Energy and Buildings,* vol. 64, pp. 90-94, 2013.

[22] F. Islam, D. Petriu and M. Woodside, "Simplifying Layered Queuing Network Models," *Beltrán M., Knottenbelt W., Bradley J. (eds) Computer Performance Engineering (EPEW 2015), Springer LNCS,* vol. 9272, 2015.

[23] M. Woodside, "Tutorial Introduction to Layered Modeling of Software Performance," February 2013. [Online]. Available:

http://www.sce.carleton.ca/rads/lqns/lqn-documentation/tutorialh.pdf. [Accessed 22 December 2018].

[24] D. Fisher, J. O'Madadhain and S. White, "JUNG: Java Universal Network/Graph Framework," 5 July 2015. [Online]. Available: https://github.com/jrtom/jung. [Accessed 16 December 2018].

[25] Y. Fang, C. Nokleberg and D. Hughes, "json-simple," 13 April 2015. [Online]. Available: https://code.google.com/archive/p/json-simple. [Accessed 16 December 2018].

[26] Google, "Guava," 29 May 2014. [Online]. Available: https://github.com/google/guava. [Accessed 15 December 2018].

[27] Apache Commons, "Commons Math: The Apache Commons Mathematics Library," [Online]. Available: http://commons.apache.org/proper/commons-math/ . [Accessed 17 December 2018].

[28] Eclipse Foundation, "Eclipse IDE," [Online]. Available: https://www.eclipse.org/ide/. [Accessed 16 December 2018].

[29] AdoptOpenJDK, "Prebuilt OpenJDK Binaries," [Online]. Available: https://adoptopenjdk.net/?variant=openjdk8&jvmVariant=openj9 . [Accessed 17 December 2018].

[30] Canonical, "Ubuntu 16.04.5 LTS (Xenial Xerus)," [Online]. Available: http://releases.ubuntu.com/16.04/. [Accessed 17 December 2018].

[31] The Kubernetes Authors, "Kubernetes Basics," The Linux Foundation, [Online]. Available: https://kubernetes.io/docs/tutorials/kubernetes-basics/. [Accessed 18

December 2018].

[32] Rancher Labs Inc., "Why Rancher?," [Online]. Available: https://rancher.com/what-is-rancher/overview/. [Accessed 10 March 2019].

Appendices

Appendix A Cloud Structure, Resources and Server Details

Table 15 – Delay (milliseconds) between entities (user/clouds)

Delay	User	Cloud Edge	Cloud Small	Cloud Medium	Cloud Large
User	N/A	25	100	175	250
Cloud Edge	25	N/A	75	150	225
Cloud Small	100	75	N/A	75	150
Cloud Medium	175	150	75	N/A	75
Cloud Large	250	225	150	75	N/A

Table 16 – Overall processing and memory capacity of the clouds

	Total memory capacity [GB]	Total processing capacity [ssj_ops]	Total number of CPUs
Cloud Edge	64.0	1,669,696.80	32
Cloud Small	80.0	2,028,905.60	40
Cloud Medium	176.0	4,240,154.40	88
Cloud Large	272.0	7,467,443.20	200

Table 17 – Server inventory for Cloud Edge

Server Name	Number of Available Servers	Total Quantity
Fujitsu Server PRIMERGY RX1330 M1	2	2
Fujitsu Server PRIMERGY TX1330 M2	1	1
Fujitsu Server PRIMERGY RX1330 M3	1	1

Table 18 – Server inventory for Cloud Small

Server Name	Number of Available Servers	Total Quantity
Fujitsu Server PRIMERGY TX1320 M2	2	3
Fujitsu Server PRIMERGY RX1330 M1	1	3
Fujitsu Server PRIMERGY TX1330 M2	1	3
Fujitsu Server PRIMERGY RX1330 M3	1	3

Table 19 – Server inventory for Cloud Medium

Server Name	Number of Available Servers	Total Quantity
Fujitsu Server PRIMERGY TX1320 M2	2	5
Fujitsu Server PRIMERGY RX1330 M1	1	5
Fujitsu Server PRIMERGY TX1330 M2	4	5
Fujitsu Server PRIMERGY RX1330 M3	0	5
QuantaGrid S31A-1U	4	5
Inspur Corporation NF5280 M4	0	5

Table 20 – Server inventory for Cloud Large

Server Name	Number of Available Servers	Total Quantity
Fujitsu Server PRIMERGY TX1320 M2	1	10
Fujitsu Server PRIMERGY RX1330 M1	1	10
Fujitsu Server PRIMERGY TX1330 M2	1	10
Fujitsu Server PRIMERGY RX1330 M3	1	10
QuantaGrid S31A-1U	1	10
Inspur Corporation NF5280 M4	1	10
Dell Inc. PowerEdge R630	1	10
Dell Inc. PowerEdge R740	0	10

Table 21 – Server details

Server Name	Link to the SPECpower_ssj2008 Results
Fujitsu Server PRIMERGY TX1320 M2	https://www.spec.org/power_ssj2008/results/res2015q4/power_ssj2008-20151110-00704.txt
Fujitsu Server PRIMERGY RX1330 M1	https://www.spec.org/power_ssj2008/results/res2015q2/power_ssj2008-20150512-00693.txt
Fujitsu Server PRIMERGY TX1330 M2	https://www.spec.org/power_ssj2008/results/res2016q1/power_ssj2008-20151214-00707.txt
Fujitsu Server PRIMERGY RX1330 M3	https://www.spec.org/power_ssj2008/results/res2017q2/power_ssj2008-20170315-00744.txt
QuantaGrid S31A-1U	https://www.spec.org/power_ssj2008/results/res2016q2/power_ssj2008-20160509-00729.txt
Inspur Corporation NF5280 M4	https://www.spec.org/power_ssj2008/results/res2017q3/power_ssj2008-20170807-00775.txt
Dell Inc. PowerEdge R630	https://www.spec.org/power_ssj2008/results/res2015q2/power_ssj2008-20150317-00691.txt
Dell Inc. PowerEdge R740	https://www.spec.org/power_ssj2008/results/res2017q3/power_ssj2008-20170829-00780.txt

Appendix B LQN Models for Tuning Analysis

Refer to Figure 20 for tuning model 1.

Download links:

LQN model - https://drive.google.com/open?id=1IhZQbricqzcysvG2a5z9DapjSLreJtL0

LQN .ps file - https://drive.google.com/open?id=1mhA_9NHKw-nBRqHHXxw1m4DiDUFtD36M

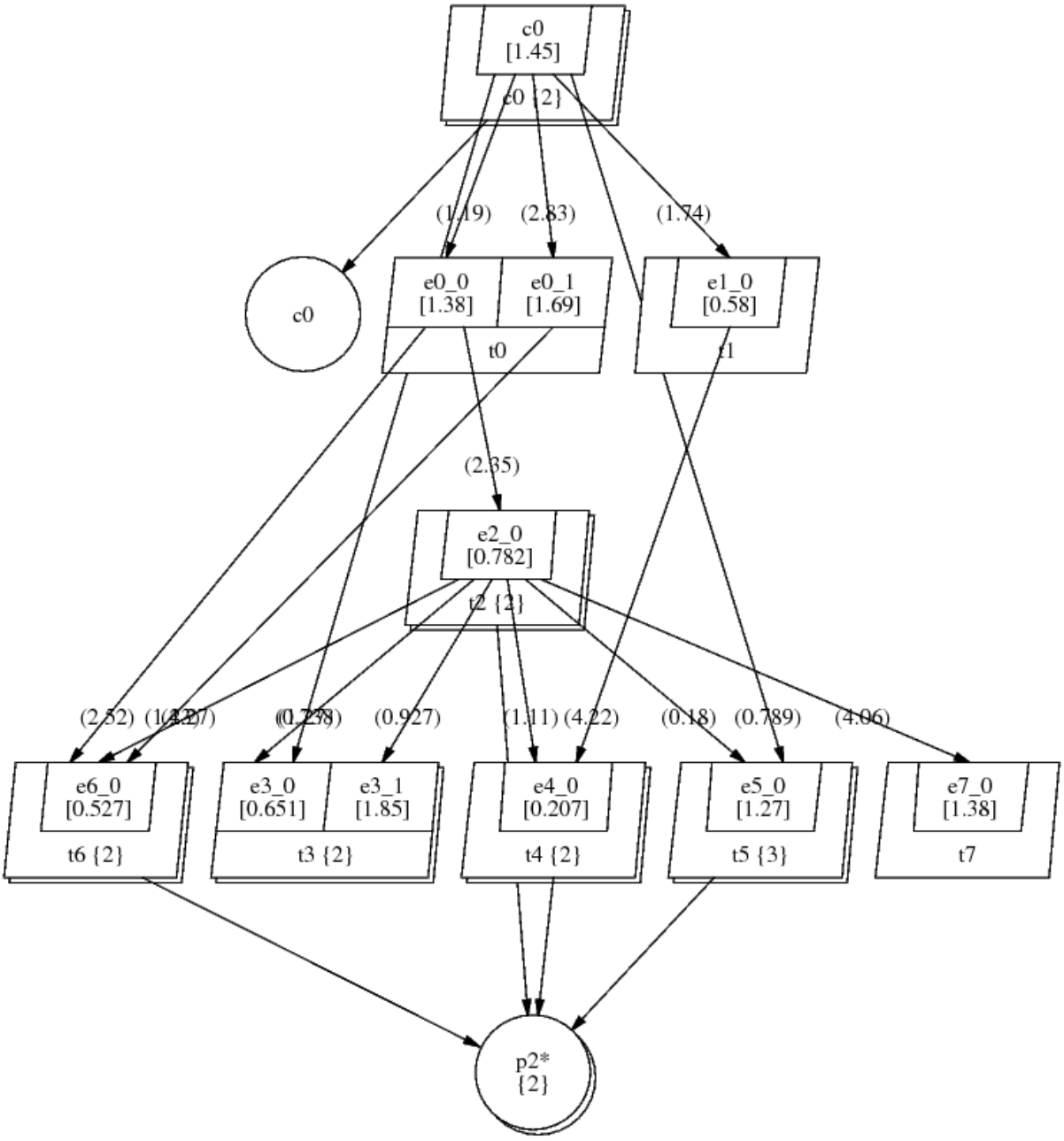

Figure 27 – Tuning Model 2 (-A8)

Download links:

LQN model - https://drive.google.com/open?id=1a1zNVLDIWNi9d7Vc0VUhUIXtGOMh_jEU

LQN .ps file - https://drive.google.com/open?id=1O8E_ifndJsog90nuVi0QPGYkNKvx_6vp

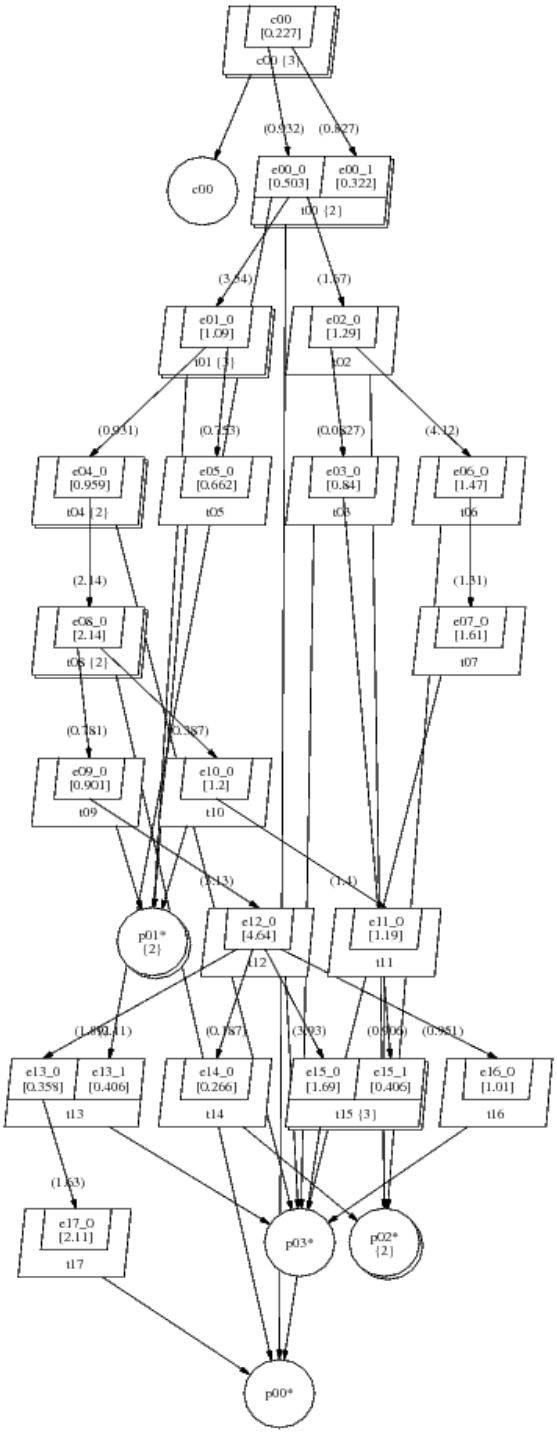

Figure 28 – Tuning Model 3 (-A18)

Download links:

LQN model - https://drive.google.com/open?id=1GdqUBcALRcq8IAQS27ImAdGZGhKB1Hlj

LQN .ps file - https://drive.google.com/open?id=1shhrnki6EAnu80-d-NjWQyLqYa_zWFsg

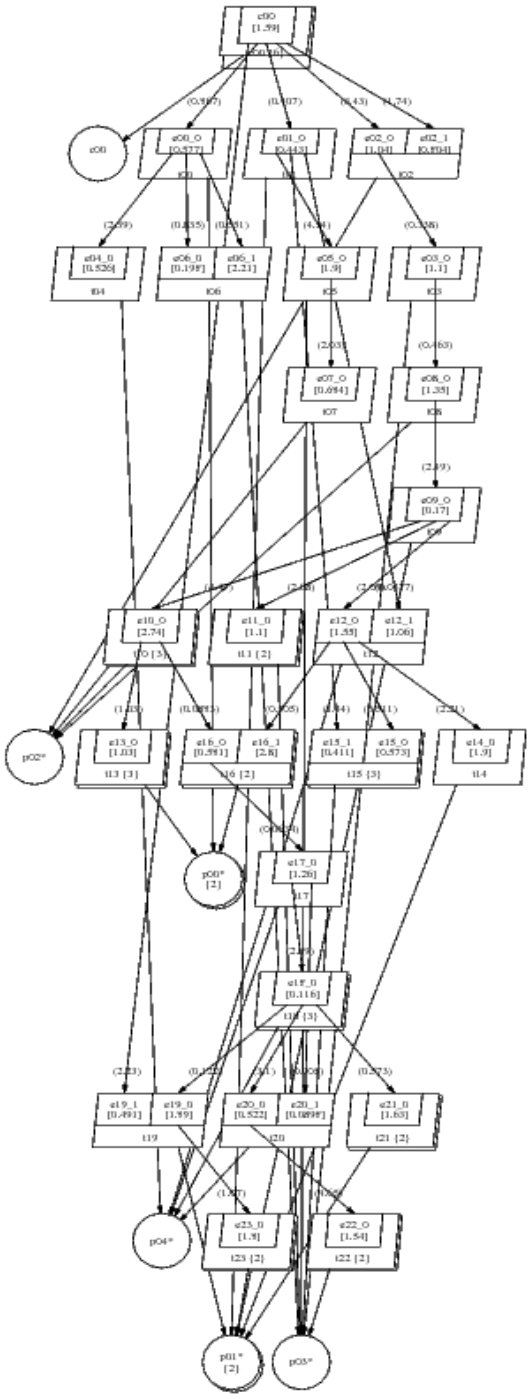

Figure 29 – Tuning Model 4 (-A24)

Download links:

LQN model - https://drive.google.com/open?id=195PUq-swnDZzSN2IWq_NfA3_FQVKQTWh

LQN .ps file - https://drive.google.com/open?id=1pOvrvR6JkZjjhu2kFY9SKdjR1Ih-zxMb

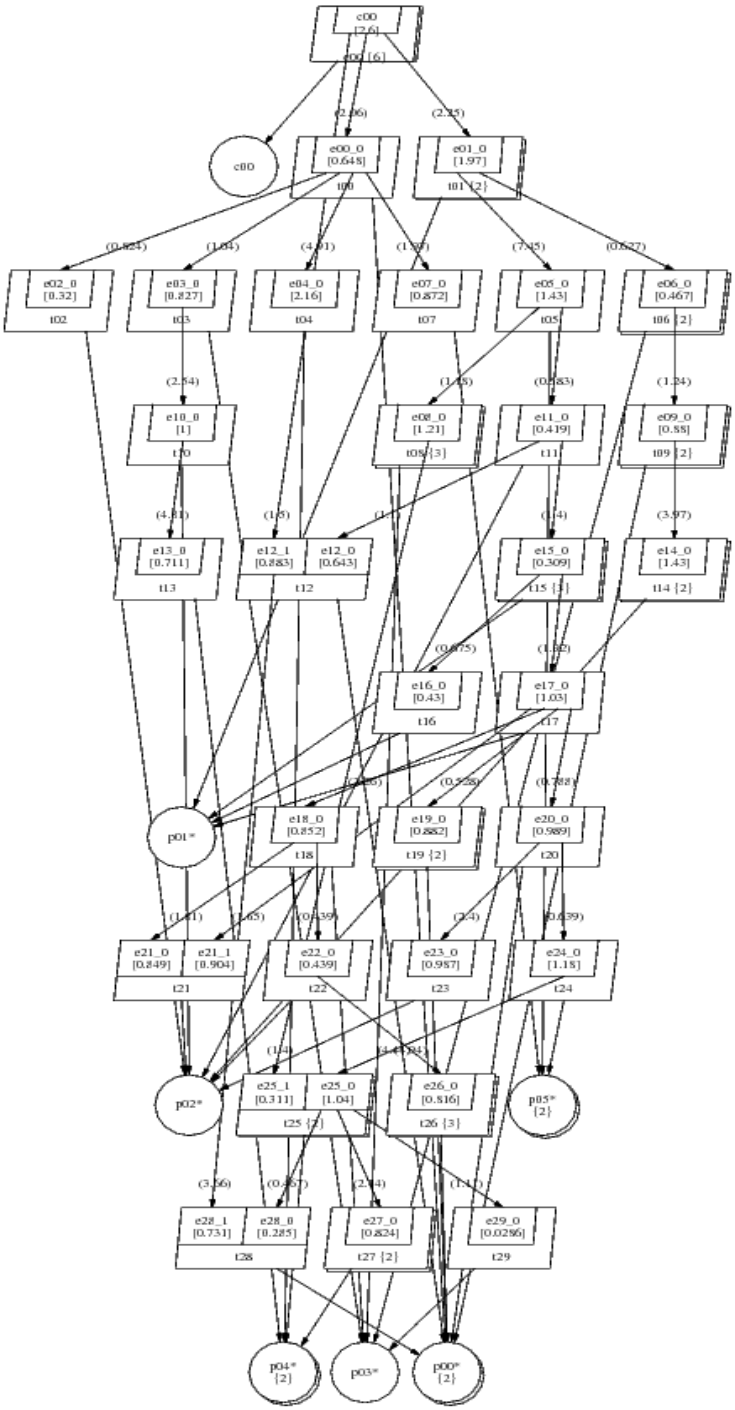

Figure 30 – Tuning Model 5 (-A30)

Download links:

LQN model - https://drive.google.com/open?id=1AOp0qsHLtZ_OezhiPO2HBxi69s8aEz45

LQN .ps file - https://drive.google.com/open?id=19efw0IzLQv6hDU7wRfx9d1MMzI5HYMRH

Appendix C Summary of Cases for Tuning Analysis

ms is the abbreviation for milliseconds.

C.1 Tuning Model 1

Table 22 – Vary throughput setup for tuning model 1

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	1.0	All LQN tasks	No limit	1.0	1.0	2000
2	2.0	All LQN tasks	No limit	1.0	1.0	2000
3	4.0	All LQN tasks	No limit	1.0	1.0	2000
4	8.0	All LQN tasks	No limit	1.0	1.0	2000
5	12.0	All LQN tasks	No limit	1.0	1.0	2000
6	16.0	All LQN tasks	No limit	1.0	1.0	2000
7	20.0	All LQN tasks	No limit	1.0	1.0	2000

Table 23 – Application characteristics for tuning model 1 based upon settings in Table 22

Case	Number of nodes with coarsening, $N_{coarsen}^{nodes}$	Number of nodes without coarsening, $N_{no-coarsen}^{nodes}$	Memory, M_{total} [GB]	Processing Capacity, P_{total} [$ssj\ ops$]
1	10	12	27.23	674,432.75
2	17	21	48.94	1,217,155.50
3	31	39	92.35	2,302,600.99
4	61	76	182.85	4,564,959.07
5	89	112	273.07	6,820,586.59
6	118	148	359.57	8,982,944.67
7	148	185	450.06	11,245,302.76

Table 24 – Vary throughput results with coarsening for tuning model 1

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	77.64	57.87	25.46	$ID_{random48}$	$D_{partition}$	380.77	121.61
2	0	132.13	104.44	20.95	$ID_{random23}$	$D_{partition}$	411.21	313.02
3	0	231.15	197.59	14.52	$ID_{random44}$	$D_{partition}$	683.64	863.40
4	0	460.13	434.55	5.56	$ID_{random6}$	$D_{partition}$	740.78	3,090.65
5	0	733.61	692.57	5.60	$ID_{random43}$	$D_{partition}$	816.00	5,216.93
6	0	996.36	949.78	4.68	$ID_{random18}$	$D_{partition}$	888.67	7,036.94
7	0	1,276.33	1,225.22	4.00	$ID_{random13}$	$D_{partition}$	709.76	11,751.72

Table 25 – Vary throughput results without coarsening for tuning model 1

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	77.63	57.87	25.45	$ID_{random15}$	$D_{partition}$	570.85	94.85
2	0	132.13	104.44	20.95	$ID_{random46}$	$D_{partition}$	877.88	217.60
3	0	217.46	197.59	9.14	$ID_{random7}$	$D_{partition}$	1,428.28	788.32
4	0	458.97	434.55	5.32	$ID_{random28}$	$D_{partition}$	1,511.73	2,593.38
5	0	734.40	692.57	5.70	$ID_{random16}$	$D_{partition}$	1,203.58	5,387.52
6	0	994.49	949.78	4.50	$ID_{random21}$	$D_{partition}$	1,174.57	9,747.09
7	0	1,270.06	1,225.22	3.53	$ID_{random26}$	$D_{partition}$	1,223.92	15,198.73

Table 26 – Vary response time constraint setup for tuning model 1

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	16.0	All LQN tasks	No limit	1.0	1.0	800
2	16.0	All LQN tasks	No limit	1.0	1.0	600
3	16.0	All LQN tasks	No limit	1.0	1.0	400
4	16.0	All LQN tasks	No limit	1.0	1.0	300
5	16.0	All LQN tasks	No limit	1.0	1.0	200
6	16.0	All LQN tasks	No limit	1.0	1.0	250

Table 27 – Vary response time constraint results with coarsening

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	996.18	949.78	4.66	$ID_{random49}$	$D_{partition}$	779.59	8,947.41
2	1	998.00	949.78	4.83	$ID_{random33}$	$D_{partition}$	566.78	13,882.92
3	6	1,000.09	949.78	5.03	$ID_{random9}$	$D_{partition}$	398.13	15,677.21
4	37	1,082.89	949.78	12.29	$ID_{random20}$	$D_{partition}$	299.84	6,633.24
5	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
6	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A

Table 28 – Vary response time constraint results without coarsening

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	1,006.16	949.78	5.60	$ID_{random31}$	$D_{partition}$	798.57	28,161.81
2	51	1,043.54	949.78	8.98	ID_{delay}	$D_{partition}$	599.64	1,547.47
3	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
4	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
5	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
6	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A

Table 29 – Vary window size setup for tuning model 1

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	16.0	1	No limit	1.0	1.0	300
2	16.0	2	No limit	1.0	1.0	300
3	16.0	4	No limit	1.0	1.0	300
4	16.0	29	No limit	1.0	1.0	300
5	16.0	58	No limit	1.0	1.0	300
6	16.0	All LQN tasks	No limit	1.0	1.0	300

Table 30 – Vary window size results with coarsening for tuning model 1

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	23	1,039.15	949.78	8.60	$ID_{random7}$	$D_{partition}$	297.90	3,942.79
2	23	1,039.15	949.78	8.60	$ID_{random24}$	$D_{partition}$	297.90	4,525.13
3	17	1,038.39	949.78	8.53	$ID_{random12}$	$D_{partition}$	299.83	6,590.96
4	49	1,041.61	949.78	8.82	$ID_{random44}$	$D_{partition}$	299.83	3,262.66
5	43	1,083.92	949.78	12.38	$ID_{random12}$	$D_{partition}$	299.84	11,719.91
6	33	1,075.18	949.78	11.66	$ID_{random36}$	$D_{partition}$	299.56	8,362.50

Table 31 – Vary window size results without coarsening for tuning model 1

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
2	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
3	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
4	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
5	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A
6	52	N/A	949.78	N/A	N/A	N/A	N/A	N/A

C.2 Tuning Model 2

Table 32 – Vary throughput setup for tuning model 2

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	0.1	All LQN tasks	No limit	1.0	1.0	50000
2	0.3	All LQN tasks	No limit	1.0	1.0	50000
3	0.5	All LQN tasks	No limit	1.0	1.0	50000
4	0.8	All LQN tasks	No limit	1.0	1.0	50000
5	1.0	All LQN tasks	No limit	1.0	1.0	50000
6	1.2	All LQN tasks	No limit	1.0	1.0	50000
7	1.3	All LQN tasks	No limit	1.0	1.0	50000

Table 33 - Application characteristics for tuning model 2 based upon settings in Table 32

Case	Number of nodes with coarsening, $N_{coarsen}^{nodes}$	Number of nodes without coarsening, $N_{no-coarsen}^{nodes}$	Memory, M_{total} [GB]	Processing Capacity, P_{total} [$ssj\ ops$]
1	19	20	48.72	1,211,691.06
2	46	50	128.97	3,217,919.98
3	75	81	210.00	5,243,701.46
4	118	157	334.20	8,348,644.74
5	145	157	411.25	10,275,116.96
6	174	188	493.60	12,333,795.16
7	188	203	534.00	13,343,729.62

Table 34 – Vary throughput results with coarsening for tuning model 2

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	131.89	103.98	21.17	$ID_{random4}$	$D_{partition}$	2,562.53	115.86
2	0	328.60	285.33	13.17	$ID_{random34}$	$D_{partition}$	15,102.45	821.82
3	0	541.14	509.74	5.80	$ID_{random39}$	$D_{partition}$	13,156.41	2,309.48
4	0	915.28	873.54	4.56	$ID_{random45}$	$D_{partition}$	13,457.91	4,171.91
5	0	1,156.54	1,107.10	4.27	$ID_{random27}$	$D_{partition}$	11,912.32	7,571.76
6	0	1,413.22	1,361.46	3.66	$ID_{random6}$	$D_{partition}$	10,721.10	7,676.93
7	2	1,560.48	1,488.80	4.59	$ID_{random14}$	$D_{partition}$	10,870.66	6,322.18

Table 35 – Vary throughput results without coarsening for tuning model 2

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	140.00	103.98	25.73	$ID_{random28}$	$D_{partition}$	10,070.61	94.52
2	0	315.35	285.33	9.52	$ID_{random30}$	$D_{partition}$	17,438.10	873.30
3	0	546.22	509.74	6.68	$ID_{random38}$	$D_{partition}$	17,146.17	2,390.10
4	0	1,152.26	1,107.10	3.92	$ID_{random36}$	$D_{partition}$	13,318.61	8,781.88
5	0	1,153.06	1,107.10	3.99	$ID_{random21}$	$D_{partition}$	12,854.53	9,241.36
6	0	1,408.98	1,361.46	3.37	$ID_{random22}$	$D_{partition}$	13,308.76	10,664.39
7	2	1,557.55	1,488.80	4.41	$ID_{random45}$	$D_{partition}$	14,089.23	10,005.69

Table 36 – Vary response time constraint setup for tuning model 2

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	1.0	All LQN tasks	No limit	1.0	1.0	12500
2	1.0	All LQN tasks	No limit	1.0	1.0	10000
3	1.0	All LQN tasks	No limit	1.0	1.0	7500
4	1.0	All LQN tasks	No limit	1.0	1.0	5000
5	1.0	All LQN tasks	No limit	1.0	1.0	2500

Table 37 – Vary response time constraint results with coarsening for tuning model 2

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	1	1,155.19	1,107.10	4.16	$ID_{random29}$	$D_{partition}$	12,454.90	7,983.02
2	1	1,158.57	1,107.10	4.44	$ID_{random18}$	$D_{partition}$	9,954.82	9,865.32
3	1	1,151.83	1,107.10	3.88	$ID_{random48}$	$D_{partition}$	7,446.08	18,654.13
4	38	1,160.44	1,107.10	4.60	$ID_{random10}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	4,988.93	10,460.56
5	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A

Table 38 – Vary response time constraint results without coarsening for tuning model 2

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	1	1,153.24	1,107.10	4.00	$ID_{random43}$	$D_{partition}$	12,063.76	10,770.10
2	2	1,154.93	1,107.10	4.14	$ID_{random36}$	$D_{partition}$	9,751.70	15,645.28
3	25	1,153.73	1,107.10	4.04	$ID_{random26}$	$D_{partition}$	7,384.49	13,574.79
4	50	1,163.67	1,107.10	4.86	$ID_{random31}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	4,974.96	2,363.79
5	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A

Table 39 – Vary window size setup for tuning model 2

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	1.0	1	No limit	1.0	1.0	5000
2	1.0	2	No limit	1.0	1.0	5000
3	1.0	4	No limit	1.0	1.0	5000
4	1.0	36	No limit	1.0	1.0	5000
5	1.0	72	No limit	1.0	1.0	5000
6	1.0	All LQN tasks	No limit	1.0	1.0	5000

Table 40 – Vary window size results with coarsening for tuning model 2

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A
2	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A
3	51	1,172.02	1,107.10	5.54	$ID_{random29}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	4,987.85	1,231.06
4	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A
5	43	1,152.42	1,107.10	3.93	$ID_{random8}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	4,984.37	4,105.23
6	36	1,153.66	1,107.10	4.04	$ID_{random48}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	4,996.70	11,791.63

Table 41 – Vary window size results without coarsening for tuning model 2

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	51	1,176.84	1,107.10	5.93	$ID_{random4}$	$D_{partition}$	4,987.58	900.61
2	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A
3	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A
4	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A
5	52	N/A	1,107.10	N/A	N/A	N/A	N/A	N/A
6	48	1,161.38	1,107.10	4.67	$ID_{random13}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	4,962.51	3,931.70

C.3 Tuning Model 3

Table 42 – Vary throughput setup for tuning model 3

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	0.01	All LQN tasks	No limit	1.0	1.0	100000
2	0.05	All LQN tasks	No limit	1.0	1.0	100000
3	0.10	All LQN tasks	No limit	1.0	1.0	100000
4	0.15	All LQN tasks	No limit	1.0	1.0	100000
5	0.20	All LQN tasks	No limit	1.0	1.0	100000
6	0.25	All LQN tasks	No limit	1.0	1.0	100000

Table 43 – Application characteristics for tuning model 3 based upon settings in Table 42

Case	Number of nodes with coarsening, $N_{coarsen}^{nodes}$	Number of nodes without coarsening, $N_{no-coarsen}^{nodes}$	Memory, M_{total} [GB]	Processing Capacity, P_{total} [$ssj\ ops$]
1	22	24	31.22	773,693.48
2	49	57	98.91	2,465,968.65
3	90	101	186.94	4,666,771.22
4	128	145	273.06	6,819,695.43
5	169	189	361.09	9,020,498.00
6	210	233	448.31	11,200,885.30

Table 44 – Vary throughput results with coarsening for tuning model 3

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	90.18	66.39	26.38	$ID_{random35}$	$D_{partition}$	24,331.90	165.06
2	0	254.29	211.61	16.79	$ID_{random19}$	$D_{partition}$	24,893.43	1,432.27
3	0	476.22	445.83	6.38	$ID_{random16}$	$D_{partition}$	34,597.62	4,670.16
4	0	733.94	692.46	5.65	$ID_{random23}$	$D_{partition}$	36,945.36	7,970.71
5	0	996.21	954.35	4.20	$ID_{random11}$	$D_{partition}$	36,451.27	15,717.87
6	0	1,270.92	1,219.81	4.02	$ID_{random49}$	$D_{partition}$	32,392.05	26,231.49

Table 45 – Vary throughput results without coarsening for tuning model 3

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	90.18	66.39	26.38	$ID_{random34}$	$D_{partition}$	44,725.34	177.39
2	0	254.30	211.61	16.79	$ID_{random24}$	$D_{partition}$	35,464.41	1,355.52
3	0	478.12	445.83	6.75	$ID_{random17}$	$D_{partition}$	48,049.49	4,726.84
4	0	735.01	692.46	5.79	$ID_{random5}$	$D_{partition}$	44,201.24	9,173.14
5	0	999.69	954.35	4.54	$ID_{random31}$	$D_{partition}$	39,405.02	18,387.34
6	0	1,264.92	1,219.81	3.57	$ID_{random20}$	$D_{partition}$	34,700.93	33,693.86

Table 46 – Vary response time constraint setup for tuning model 3

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	0.20	All LQN tasks	No limit	1.0	1.0	35000
2	0.20	All LQN tasks	No limit	1.0	1.0	25000
3	0.20	All LQN tasks	No limit	1.0	1.0	15000
4	0.20	All LQN tasks	No limit	1.0	1.0	10000
5	0.20	All LQN tasks	No limit	1.0	1.0	5000

Table 47 – Vary response time constraint results with coarsening for tuning model 3

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	997.20	954.35	4.30	$ID_{random33}$	$D_{partition}$	33,466.64	16,937.44
2	0	1,018.90	954.35	6.34	$ID_{random42}$	$D_{partition}$	24,917.89	36,008.60
3	45	1,004.72	954.35	5.01	$ID_{random42}$	$D_{partition}$	14,933.73	14,328.99
4	31	1,008.69	954.35	5.39	$ID_{random7}$	$D_{partition}$	9,934.00	54,464.25
5	52	N/A	954.35	N/A	N/A	N/A	N/A	N/A

Table 48 – Vary response time constraint results without coarsening for tuning model 3

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	1,005.92	954.35	5.13	$ID_{random8}$	$D_{partition}$	34,954.71	26,184.05
2	0	1,013.19	954.35	5.81	$ID_{random11}$	$D_{partition}$	24,895.63	45,342.12
3	48	1,005.53	954.35	5.09	$ID_{random5}$	$D_{partition}$	14,969.66	9,642.07
4	50	1,009.06	954.35	5.42	$ID_{random27}$	$D_{BP\text{-}RUAEE\text{-}Coarsen}$	9,884.02	7,189.53
5	52	N/A	954.35	N/A	N/A	N/A	N/A	N/A

Table 49 – Vary window size setup for tuning model 3

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	0.20	1	No limit	1.0	1.0	10000
2	0.20	2	No limit	1.0	1.0	10000
3	0.20	4	No limit	1.0	1.0	10000
4	0.20	42	No limit	1.0	1.0	10000
5	0.20	84	No limit	1.0	1.0	10000
6	0.20	All LQN tasks	No limit	1.0	1.0	10000

Table 50 – Vary window size results with coarsening for tuning model 3

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	51	1,012.29	954.35	5.72	ID_{power}	$D_{partition}$	9,930.86	2,313.81
2	47	1,007.00	954.35	5.23	ID_{power}	$D_{partition}$	9,967.52	3,086.29
3	40	1,004.46	954.35	4.99	$ID_{random15}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	9,970.40	4,343.68
4	15	1,002.82	954.35	4.83	$ID_{random31}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	9,877.03	29,614.03
5	33	1,004.15	954.35	4.96	$ID_{random20}$	$D_{BP\text{-}HBF\text{-}Proc\text{-}Coarsen}$	9,911.95	27,560.43
6	33	1,008.73	954.35	5.39	$ID_{random18}$	$D_{partition}$	9,952.18	47,539.06

Table 51 – Vary window size results without coarsening for tuning model 3

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	46	1,007.13	954.35	5.24	$ID_{random40}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	9,846.03	2,724.85
2	46	998.88	954.35	4.46	$ID_{random38}$	$D_{partition}$	9,973.87	3,031.09
3	39	999.50	954.35	4.52	ID_{power}	$D_{partition}$	9,971.79	4,509.31
4	33	1,007.38	954.35	5.26	$ID_{random47}$	$D_{BP\text{-}RUAEE\text{-}Coarsen}$	9,999.75	31,680.77
5	49	1,009.39	954.35	5.45	$ID_{random33}$	$D_{BP\text{-}RUAEE\text{-}Coarsen}$	9,916.58	8,514.60
6	51	1,027.99	954.35	7.16	ID_{power}	$D_{partition}$	9,874.08	4,427.75

C.4 Tuning Model 4

Table 52 – Vary throughput setup for tuning model 4

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	0.05	All LQN tasks	No limit	1.0	1.0	15000
2	0.10	All LQN tasks	No limit	1.0	1.0	15000
3	0.50	All LQN tasks	No limit	1.0	1.0	15000
4	1.00	All LQN tasks	No limit	1.0	1.0	15000
5	1.25	All LQN tasks	No limit	1.0	1.0	15000
6	1.40	All LQN tasks	No limit	1.0	1.0	15000

Table 53 – Application characteristics for tuning model 4 based upon settings in Table 52

Case	Number of nodes with coarsening, $N_{coarsen}^{nodes}$	Number of nodes without coarsening, $N_{no-coarsen}^{nodes}$	Memory, M_{total} [GB]	Processing Capacity, P_{total} [$ssj\ ops$]
1	23	27	47.05	1,168,047.84
2	25	32	61.58	1,531,289.38
3	74	82	198.23	4,947,492.79
4	134	149	372.31	9,299,365.47
5	162	177	451.45	11,277,939.73
6	182	202	511.60	12,781,754.47

Table 54 – Vary throughput results with coarsening for tuning model 4

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	134.86	100.23	25.68	$ID_{random38}$	$D_{partition}$	3,191.62	231.88
2	0	146.31	131.40	10.19	$ID_{random41}$	$D_{partition}$	1,409.25	535.14
3	0	504.08	476.93	5.39	$ID_{random8}$	$D_{partition}$	5,962.44	2,006.75
4	0	1,032.14	988.30	4.25	$ID_{random29}$	$D_{partition}$	6,829.58	6,080.89
5	0	1,284.52	1,229.19	4.31	$ID_{random6}$	$D_{partition}$	5,903.52	7,092.00
6	0	1,477.85	1,417.94	4.05	$ID_{random22}$	$D_{partition}$	6,187.15	8,458.38

Table 55 – Vary throughput results without coarsening for tuning model 4

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	134.86	100.23	25.68	ID_{power}	$D_{partition}$	5,887.97	153.13
2	0	162.42	131.40	19.10	$ID_{random3}$	$D_{partition}$	7,883.81	322.23
3	0	513.41	476.93	7.11	$ID_{random46}$	$D_{partition}$	7,932.24	1,952.57
4	0	1,031.80	988.30	4.22	$ID_{random20}$	$D_{partition}$	6,837.31	6,689.24
5	0	1,279.91	1,229.19	3.96	$ID_{random23}$	$D_{partition}$	6,962.30	9,354.91
6	0	1,466.09	1,417.94	3.28	$ID_{random10}$	$D_{partition}$	9,519.68	11,163.74

Table 56 – Vary response time constraint setup for tuning model 4

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	1.25	All LQN tasks	No limit	1.0	1.0	8000
2	1.25	All LQN tasks	No limit	1.0	1.0	5000
3	1.25	All LQN tasks	No limit	1.0	1.0	2000
4	1.25	All LQN tasks	No limit	1.0	1.0	1000
5	1.25	All LQN tasks	No limit	1.0	1.0	3000

Table 57 – Vary response time constraint results with coarsening for tuning model 4

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	1,280.36	1,229.19	4.00	$ID_{random50}$	$D_{partition}$	5,470.73	6,833.44
2	0	1,280.95	1,229.19	4.04	$ID_{random11}$	$D_{partition}$	4,995.82	9,181.86
3	27	1,290.89	1,229.19	4.78	ID_{power}	$D_{partition}$	1,987.91	21,513.02
4	52	N/A	1,229.19	N/A	N/A	N/A	N/A	N/A
5	0	1,273.24	1,229.19	3.46	$ID_{random16}$	$D_{partition}$	2,996.16	25,175.39

Table 58 – Vary response time constraint results without coarsening for tuning model 4

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	1,276.93	1,229.19	3.74	$ID_{random47}$	$D_{partition}$	7,210.03	9,577.46
2	0	1,273.99	1,229.19	3.52	$ID_{random14}$	$D_{partition}$	4,741.07	19,653.12
3	47	1,306.98	1,229.19	5.95	$ID_{random2}$	$D_{partition}$	1,984.42	7,362.07
4	52	N/A	1,229.19	N/A	N/A	N/A	N/A	N/A
5	1	1,284.27	1,229.19	4.29	$ID_{random10}$	$D_{partition}$	2,988.32	44,156.25

Table 59 – Vary window size setup for tuning model 4

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	1.25	1	No limit	1.0	1.0	2000
2	1.25	2	No limit	1.0	1.0	2000
3	1.25	4	No limit	1.0	1.0	2000
4	1.25	40	No limit	1.0	1.0	2000
5	1.25	80	No limit	1.0	1.0	2000
6	1.25	All LQN tasks	No limit	1.0	1.0	2000

Table 60 – Vary window size results with coarsening for tuning model 4

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	22	1,281.38	1,229.19	4.07	$ID_{random8}$	$D_{partition}$	1,980.48	3,421.16
2	27	1,279.45	1,229.19	3.93	$ID_{random15}$	$D_{partition}$	1,986.25	3,612.80
3	25	1,274.27	1,229.19	3.54	$ID_{random10}$	$D_{partition}$	1,997.83	5,021.07
4	33	1,276.43	1,229.19	3.70	ID_{power}	$D_{partition}$	1,947.35	21,410.82
5	29	1,283.52	1,229.19	4.23	ID_{power}	$D_{partition}$	1,984.50	44,549.34
6	31	1,290.89	1,229.19	4.78	ID_{power}	$D_{partition}$	1,987.91	17,526.30

Table 61 – Vary window size results without coarsening for tuning model 4

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	48	1,296.26	1,229.19	5.17	$ID_{random29}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	1,995.93	1,538.25
2	44	1,295.25	1,229.19	5.10	$ID_{random26}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	1,999.70	1,915.12
3	41	1,299.64	1,229.19	5.42	$ID_{random11}$	$D_{partition}$	1,998.54	2,502.89
4	45	1,305.17	1,229.19	5.82	$ID_{random23}$	$D_{partition}$	1,994.60	4,178.73
5	44	1,299.15	1,229.19	5.39	$ID_{random14}$	$D_{partition}$	1,999.26	6,945.39
6	45	1,312.77	1,229.19	6.37	$ID_{random34}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	1,999.01	10,216.79

C.5 Tuning Model 5

Table 62 – Vary throughput setup for tuning model 5

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	0.01	All LQN tasks	No limit	1.0	1.0	150000
2	0.04	All LQN tasks	No limit	1.0	1.0	150000
3	0.08	All LQN tasks	No limit	1.0	1.0	150000
4	0.10	All LQN tasks	No limit	1.0	1.0	150000
5	0.20	All LQN tasks	No limit	1.0	1.0	150000
6	0.25	All LQN tasks	No limit	1.0	1.0	150000

Table 63 – Application characteristics for tuning model 5 based upon settings in Table 62

Case	Number of nodes with coarsening, $N_{coarsen}^{nodes}$	Number of nodes without coarsening, $N_{no-coarsen}^{nodes}$	Memory, M_{total} [GB]	Processing Capacity, P_{total} [$ssj\ ops$]
1	29	34	53.86	1,335,286.83
2	53	63	103.88	2,585,733.27
3	90	105	173.60	4,328,766.04
4	108	127	208.56	5,202,857.96
5	202	234	391.06	9,765,436.32
6	251	289	483.34	12,072,434.36

Table 64 – Vary throughput results with coarsening for tuning model 5

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	137.31	114.58	16.55	$ID_{random5}$	$D_{partition}$	5,811.35	396.04
2	0	261.18	221.88	15.05	$ID_{random26}$	$D_{partition}$	72,235.59	1,338.08
3	0	439.11	408.39	7.00	$ID_{random2}$	$D_{partition}$	87,783.89	3,305.79
4	0	541.56	505.22	6.71	$ID_{random20}$	$D_{partition}$	100,236.27	4,761.93
5	0	1,093.92	1,045.05	4.47	$ID_{random8}$	$D_{partition}$	89,933.90	25,437.10
6	0	1,379.51	1,328.51	3.70	$ID_{random28}$	$D_{partition}$	85,573.95	52,476.93

Table 65 – Vary throughput results without coarsening for tuning model 5

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	137.31	114.58	16.55	$ID_{random10}$	$D_{partition}$	5,811.35	305.98
2	0	262.80	221.88	15.57	$ID_{random15}$	$D_{partition}$	97,087.41	1,201.61
3	0	437.21	408.39	6.59	$ID_{random34}$	$D_{partition}$	101,333.49	4,232.57
4	0	542.99	505.22	6.96	$ID_{random45}$	$D_{partition}$	90,597.51	6,597.84
5	0	1,088.58	1,045.05	4.00	$ID_{random35}$	$D_{partition}$	98,409.72	31,381.37
6	0	1,378.86	1,328.51	3.65	$ID_{random38}$	$D_{partition}$	98,912.77	62,772.60

Table 66 – Vary response time constraint setup for tuning model 5

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	0.10	All LQN tasks	No limit	1.0	1.0	60000
2	0.10	All LQN tasks	No limit	1.0	1.0	50000
3	0.10	All LQN tasks	No limit	1.0	1.0	40000
4	0.10	All LQN tasks	No limit	1.0	1.0	30000
5	0.10	All LQN tasks	No limit	1.0	1.0	20000
6	0.10	All LQN tasks	No limit	1.0	1.0	10000
7	0.10	All LQN tasks	No limit	1.0	1.0	5000
8	0.10	All LQN tasks	No limit	1.0	1.0	7500

Table 67 – Vary response time constraint results with coarsening for tuning model 5

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	556.15	505.22	9.16	$ID_{random47}$	$D_{partition}$	58,750.58	8,063.80
2	0	554.91	505.22	8.96	$ID_{random50}$	$D_{partition}$	49,885.66	8,696.50
3	1	559.59	505.22	9.72	$ID_{random7}$	$D_{partition}$	39,422.96	9,492.19
4	3	565.00	505.22	10.58	$ID_{random48}$	$D_{partition}$	29,871.55	10,353.56
5	6	558.18	505.22	9.49	$ID_{random46}$	$D_{partition}$	19,916.44	10,479.52
6	37	556.88	505.22	9.28	$ID_{random23}$	$D_{partition}$	9,949.04	3,977.05
7	52	N/A	505.22	N/A	N/A	N/A	N/A	N/A
8	50	553.22	505.22	8.68	$ID_{random3}$	$D_{partition}$	7,376.08	1,006.91

Table 68 – Vary response time constraint results without coarsening for tuning model 5

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	0	550.18	505.22	8.17	$ID_{random18}$	$D_{partition}$	59,691.50	12,383.09
2	1	561.54	505.22	10.03	$ID_{random15}$	$D_{partition}$	49,727.40	13,669.69
3	2	558.98	505.22	9.62	$ID_{random14}$	$D_{partition}$	39,838.00	14,757.61
4	1	578.75	505.22	12.71	$ID_{random24}$	$D_{partition}$	29,940.77	16,137.77
5	7	581.82	505.22	13.17	$ID_{random21}$	$D_{partition}$	19,996.34	16,123.42
6	42	583.97	505.22	13.49	$ID_{random8}$	$D_{partition}$	9,849.15	4,218.42
7	52	N/A	505.22	N/A	N/A	N/A	N/A	N/A
8	52	N/A	505.22	N/A	N/A	N/A	N/A	N/A

Table 69 – Vary window size setup for tuning model 5

Case	Throughput, τ [requests/ms]	Window Size, P_{window}	Maximum moves per node, P_{max_moves}	Edge Weight Factor, B_{edge_factor}	Minimum Node Count, B_{node_count}	Response Time Constraint, $R_{constraint}$ [ms]
1	0.10	1	No limit	1.0	1.0	10000
2	0.10	2	No limit	1.0	1.0	10000
3	0.10	4	No limit	1.0	1.0	10000
4	0.10	26	No limit	1.0	1.0	10000
5	0.10	53	No limit	1.0	1.0	10000
6	0.10	All LQN tasks	No limit	1.0	1.0	10000

Table 70 – Vary window size results with coarsening for tuning model 5

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	48	658.92	505.22	23.33	$ID_{random41}$	$D_{BP\text{-}RUAEE\text{-}Uncoarsen}$	9,951.13	655.49
2	49	654.53	505.22	22.81	$ID_{random43}$	$D_{partition}$	9,970.35	612.15
3	45	591.65	505.22	14.61	$ID_{random43}$	$D_{partition}$	9,941.06	861.25
4	46	575.51	505.22	12.21	$ID_{random5}$	$D_{partition}$	9,758.03	1,872.86
5	39	647.60	505.22	21.99	$ID_{random7}$	$D_{partition}$	9,985.17	5,904.45
6	37	653.86	505.22	22.73	$ID_{random17}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	9,754.61	4,021.43

Table 71 – Vary window size results without coarsening for tuning model 5

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	Response Time, R_{MCAD} [ms]	MCAD Runtime, T_{MCAD} [ms]
1	47	648.95	505.22	22.15	$D_{random2}$	$D_{BP\text{-}RUAEE\text{-}Coarsen}$	9,996.83	560.69
2	47	650.11	505.22	22.29	$D_{random25}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	9,923.83	586.68
3	47	652.97	505.22	22.63	$D_{random8}$	$D_{partition}$	9,992.61	615.63
4	48	649.08	505.22	22.16	$D_{random37}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Coarsen}$	9,937.45	1,230.00
5	46	648.17	505.22	22.05	$D_{random34}$	$D_{partition}$	9,975.49	4,028.67
6	45	648.19	505.22	22.06	$D_{random45}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	9,892.12	3,207.80

Appendix D Analysis of the Tuned MCAD Algorithm

D.1 LQN Models used for Validating the Tuned MCAD Algorithm

The LQN models used for the tuning analysis can be found using the following link: "https://github.com/singhbabneet/lqn_verification/tree/master/**<MODEL_NAME>/<SOLUTION_INITIAL_DEPLOYMENT>**/delay.lqn".

The LQNS solution for an LQN model can be found using the following link: "https://github.com/singhbabneet/lqn_verification/tree/master/**<MODEL_NAME>/<SOLUTION_INITIAL_DEPLOYMENT>**/delay.out".

<MODEL_NAME> should be replaced with the model name (Table 72) and <SOLUTION_INITIAL_DEPLOYMENT> should be replaced with the subscript of the solution initial deployment (Table 72).

For case 1, the <MODEL_NAME> is **model_A4_case01** and the <SOLUTION_INITIAL_DEPLOYMENT> is $D_{random20}$.

So, the link to the LQN model for case 1 is "https://github.com/singhbabneet/lqn_verification/tree/master/**model_A4_case01/random20**/delay.lqn".

The link to the LQNS output file (results) for case 1 is "https://github.com/singhbabneet/lqn_verification/tree/master/**model_A4_case01/random20**/delay.out".

If the LQNS results do not exist, then the LQNS failed to solve the LQN model.

D.2 Data Collected for Validating the Tuned MCAD Algorithm

Table 72 – Setup information for validating the tuned MCAD algorithm

Case	Model Name	Throughput, τ [requests/ms]	Response Time Constraint, $R_{constraint}$ [ms]	Number of nodes, $N_{coarsen}^{nodes}$	Memory, M_{total} [GB]	Total Calls, Y_{total}	Processing Capacity, P_{total} $[ssj\ ops]$
1	model_A4_case01	6.00	3,000.00	96	318.67	18.24	7,960,453.81
2	model_A4_case02	6.00	3,000.00	98	298.24	15.84	7,449,826.06
3	model_A4_case04	6.00	3,000.00	80	250.54	10.64	6,257,193.51
4	model_A4_case05	10.00	1,000.00	57	202.9	6.48	5,066,334.31
5	model_A4_case06	6.00	3,000.00	118	316.27	25.67	7,900,423.51
6	model_A4_case07	6.00	3,000.00	89	258.99	8.58	6,463,106.15
7	model_A4_case08	6.00	3,000.00	93	293.03	12.73	7,319,613.81
8	model_A4_case09	10.00	1,000.00	75	206.6	5.63	5,158,725.45
9	model_A4_case10	10.00	1,000.00	48	163.18	4.13	4,073,131.26
10	model_A4_case11	8.00	2,000.00	61	138.69	5.78	3,460,930.31
11	model_A4_case12	4.00	5,000.00	160	405.32	25.89	10,126,808.96
12	model_A4_case13	10.00	1,000.00	75	189.41	7.18	4,728,982.15
13	model_A4_case14	8.00	2,000.00	77	242.17	10.88	6,048,033.44
14	model_A4_case15	4.00	5,000.00	134	442.72	19.27	11,061,669.43
15	model_A4_case16	16.00	5,000.00	73	205.18	10.01	5,111,153.45
16	model_A4_case17	4.00	5,000.00	76	271.11	15.68	6,771,585.81
17	model_A4_case18	4.00	5,000.00	86	238.99	18.36	5,968,449.53
18	model_A4_case19	16.00	5,000.00	96	206.85	3.82	5,164,969.27
19	model_A4_case20	16.00	5,000.00	36	106.8	3.55	2,663,670.63
20	model_A4_case21	10.00	1,000.00	61	222.98	6.12	5,568,283.41
21	model_A8_case02	1.00	7,500.00	58	186.42	60.84	4,654,144.91
22	model_A8_case03	1.00	7,500.00	82	190.97	96.29	4,768,085.36
23	model_A8_case04	1.00	7,500.00	103	371.57	52.33	9,282,959.75
24	model_A8_case05	1.25	7,500.00	29	66.61	23.14	1,658,912.16
25	model_A8_case06	0.75	7,500.00	95	270.02	90.55	6,744,167.93
26	model_A8_case07	0.75	7,500.00	78	256.13	61.69	6,396,998.21
27	model_A8_case08	0.50	7,500.00	71	182.83	88.06	4,564,488.60
28	model_A8_case09	0.50	7,500.00	99	156.96	159.06	3,916,276.06
29	model_A8_case10	0.50	7,500.00	60	98.15	155.32	2,447,486.60

Case	Model Name	Throughput, τ [requests/ms]	Response Time Constraint, $R_{constraint}$ [ms]	Number of nodes, $N_{coarsen}^{nodes}$	Memory, M_{total} [GB]	Total Calls, Y_{total}	Processing Capacity, P_{total} $[ssj\ ops]$
30	model_A8_case11	0.50	7,500.00	43	107.44	83.13	2,679,701.17
31	model_A8_case12	0.50	7,500.00	89	265.02	136.16	6,619,126.24
32	model_A8_case13	0.50	7,500.00	21	30.74	44.9	762,235.81
33	model_A8_case14	1.25	7,500.00	60	199.49	41.62	4,981,084.37
34	model_A8_case15	1.00	7,500.00	106	323.3	93.31	8,076,303.14
35	model_A8_case16	1.25	7,500.00	64	175.5	62.86	4,381,350.58
36	model_A8_case17	1.25	7,500.00	77	183.4	88.34	4,571,926.31
37	model_A8_case18	0.50	7,500.00	42	137.79	50.57	3,438,384.87
38	model_A8_case19	1.25	7,500.00	59	72.1	59.24	1,783,457.74
39	model_A8_case20	1.00	7,500.00	144	292.97	150.87	7,318,016.28
40	model_A8_case21	1.25	7,500.00	74	168.94	82.47	4,217,328.94
41	model_A12_case-01	0.60	10,000.00	85	178.45	136.51	4,452,504.56
42	model_A12_case-02	0.60	10,000.00	112	288.27	233.18	7,200,616.64
43	model_A12_case-03	1.00	10,000.00	96	206.69	70.68	5,157,755.05
44	model_A12_case-04	0.60	10,000.00	66	199.9	71.77	4,991,351.30
45	model_A12_case-05	0.30	20,000.00	107	353.83	253.98	8,839,622.53
46	model_A12_case-06	0.30	20,000.00	84	232.09	140.96	5,795,991.64
47	model_A12_case-07	1.00	10,000.00	128	274.79	90	6,858,709.20
48	model_A12_case-08	1.00	10,000.00	89	223.42	75.03	5,579,202.50
49	model_A12_case-09	0.60	10,000.00	102	242.63	153.5	6,059,538.92
50	model_A12_case-10	1.00	10,000.00	87	236.59	85.63	5,908,388.94
51	model_A12_case-11	0.60	10,000.00	162	305.08	281.48	7,620,723.58
52	model_A12_case-12	0.60	10,000.00	95	249.84	141.75	6,239,784.05
53	model_A12_case-13	0.30	20,000.00	87	207.52	169.67	5,172,387.20
54	model_A12_case-14	0.30	20,000.00	148	391.81	324.44	9,789,055.57
55	model_A12_case-15	0.30	20,000.00	84	283.54	201.35	7,082,353.89
56	model_A18_case01	0.10	15,000.00	115	208.48	985.77	5,202,019.05
57	model_A18_case02	0.10	15,000.00	58	155.37	388.65	3,876,906.72
58	model_A18_case03	0.10	15,000.00	91	206.65	576.91	5,159,977.31
59	model_A18_case05	0.50	15,000.00	131	406.28	131.06	10,150,785.90
60	model_A18_case07	0.05	15,000.00	75	149.34	719.16	3,724,036.95
61	model_A18_case08	0.50	15,000.00	239	518.88	386.64	12,965,838.98

Case	Model Name	Throughput, τ [requests/ms]	Response Time Constraint, $R_{constraint}$ [ms]	Number of nodes, $N_{coarsen}^{nodes}$	Memory, M_{total} [GB]	Total Calls, Y_{total}	Processing Capacity, P_{total} $[ssj\ ops]$
62	model_A18_case09	0.50	15,000.00	97	237.23	202.42	5,924,143.09
63	model_A18_case10	0.50	15,000.00	156	263.02	292.41	6,569,299.01
64	model_A18_case11	0.50	15,000.00	186	337.95	462.67	8,442,478.04
65	model_A18_case12	0.05	15,000.00	134	386.16	1,789.53	9,640,187.82
66	model_A18_case13	0.01	100,000.00	24	62.77	671.54	1,562,885.46
67	model_A18_case14	0.01	100,000.00	48	116.41	3,409.74	2,903,921.73
68	model_A18_case15	0.50	15,000.00	56	117.2	66.76	2,920,008.84
69	model_A18_case16	0.10	15,000.00	62	190.88	564.12	4,765,733.66
70	model_A18_case17	0.50	15,000.00	56	170.41	113.81	4,253,998.84
71	model_A18_case18	0.10	15,000.00	116	160.05	777.89	3,990,580.90
72	model_A18_case19	0.10	15,000.00	103	231.69	1,011.06	5,779,495.88
73	model_A18_case21	0.10	15,000.00	23	50.65	67.11	1,256,578.75
74	model_A24_case01	0.01	200,000.00	118	325.38	8,296.70	8,128,265.31
75	model_A24_case02	0.25	10,000.00	95	236.2	252.52	5,898,870.95
76	model_A24_case03	0.25	10,000.00	77	219.02	313.66	5,469,283.75
77	model_A24_case05	0.08	30,000.00	136	285.7	1,628.43	7,134,752.10
78	model_A24_case06	0.08	30,000.00	92	204.05	1,269.42	5,095,014.47
79	model_A24_case07	0.25	10,000.00	74	204.61	248.13	5,104,562.52
80	model_A24_case08	0.08	30,000.00	116	258.89	1,572.86	6,466,031.35
81	model_A24_case09	0.03	200,000.00	129	213.27	6,360.32	5,325,443.40
82	model_A24_case12	0.08	30,000.00	111	235.6	580.17	5,878,740.53
83	model_A24_case13	0.01	350,000.00	70	172.69	5,458.98	4,311,002.75
84	model_A24_case14	0.01	350,000.00	122	283.62	11,032.81	7,084,185.26
85	model_A24_case15	0.01	200,000.00	140	408.38	10,131.22	10,203,342.45
86	model_A24_case17	0.03	200,000.00	77	108.92	2,776.31	2,715,617.12
87	model_A24_case18	0.08	30,000.00	171	326.11	2,079.83	8,143,440.52
88	model_A24_case19	0.03	200,000.00	216	424.19	10,527.30	10,596,787.27
89	model_A24_case20	0.25	10,000.00	161	287.07	582.51	7,158,679.39
90	model_A24_case21	0.08	30,000.00	102	288.39	966.17	7,203,557.02
91	model_A30_case01	0.05	100,000.00	172	318.19	3,638.75	7,943,223.94
92	model_A30_case03	0.05	100,000.00	77	153.14	1,282.56	3,816,632.02
93	model_A30_case04	0.05	100,000.00	104	183.51	1,622.64	4,575,404.87

Case	Model Name	Throughput, τ [requests/ms]	Response Time Constraint, $R_{constraint}$ [ms]	Number of nodes, $N_{coarsen}^{nodes}$	Memory, M_{total} [GB]	Total Calls, Y_{total}	Processing Capacity, P_{total} $[ssj\ ops]$
94	model_A30_case05	0.05	100,000.00	179	255.87	5,144.48	6,389,761.63
95	model_A30_case06	0.10	25,000.00	50	78.17	353.35	1,938,841.47
96	model_A30_case08	0.03	150,000.00	128	335.77	3,636.95	8,387,876.30
97	model_A30_case09	0.05	100,000.00	147	370.1	2,265.11	9,246,246.07
98	model_A30_case10	0.01	150,000.00	36	48.35	1,260.12	1,195,535.04
99	model_A30_case11	0.05	100,000.00	144	342.18	2,176.98	8,548,353.52
100	model_A30_case13	0.05	100,000.00	104	200.88	1,119.81	5,007,292.09
101	model_A30_case15	0.05	100,000.00	62	157.05	665.07	3,918,627.34
102	model_A30_case16	0.05	100,000.00	109	252.17	1,611.43	6,293,574.82
103	model_A30_case18	0.03	150,000.00	40	75.62	1,347.45	1,880,921.72
104	model_A30_case20	0.01	150,000.00	122	160.69	7,429.45	4,002,427.65

Table 73 – Results for validating the tuned MCAD algorithm (Part 1)

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	MCAD Runtime, T_{MCAD} [ms]
1	0	873.82	827.20	5.34	$ID_{random20}$	$D_{partition}$	2,377.04
2	0	814.03	766.24	5.87	$ID_{random2}$	$D_{partition}$	1,192.32
3	0	662.92	626.83	5.44	$ID_{random45}$	$D_{partition}$	1,748.92
4	0	521.36	490.09	6.00	$ID_{random5}$	$D_{partition}$	1,059.91
5	0	870.07	820.03	5.75	$ID_{random15}$	$D_{partition}$	7,879.07
6	0	696.20	650.86	6.51	$ID_{random39}$	$D_{partition}$	609.93
7	0	800.77	750.79	6.24	$ID_{random21}$	$D_{partition}$	1,662.36
8	0	530.93	500.33	5.76	$ID_{random29}$	$D_{partition}$	687.75
9	0	405.55	380.07	6.28	$ID_{random5}$	$D_{partition}$	552.50
10	0	344.35	312.25	9.32	$ID_{random48}$	$D_{partition}$	3,678.80
11	0	1,135.94	1,089.04	4.13	$ID_{random49}$	$D_{partition}$	11,412.99
12	0	483.14	452.72	6.30	$ID_{random30}$	$D_{partition}$	1,027.32
13	0	639.28	602.42	5.76	$ID_{random19}$	$D_{partition}$	699.95
14	0	1,260.00	1,202.86	4.53	$ID_{random27}$	$D_{partition}$	4,397.65
15	0	525.80	495.06	5.85	$ID_{random13}$	$D_{partition}$	3,281.03
16	0	730.64	686.85	5.99	$ID_{random25}$	$D_{partition}$	933.94
17	0	635.34	593.14	6.64	$ID_{random13}$	$D_{partition}$	4,030.65
18	0	526.21	501.02	4.79	$ID_{random1}$	$D_{partition}$	8,189.40
19	0	265.87	228.57	14.03	$ID_{random30}$	$D_{partition}$	1,046.03
20	0	582.44	546.45	6.18	$ID_{random31}$	$D_{partition}$	2,299.15
21	0	474.90	444.43	6.41	$ID_{random43}$	$D_{partition}$	881.78
22	0	505.16	457.06	9.52	$ID_{random33}$	$D_{partition}$	5,146.30
23	0	1,028.37	986.31	4.09	$ID_{random22}$	$D_{partition}$	2,675.37
24	0	168.26	142.35	15.40	$ID_{random22}$	$D_{partition}$	401.27
25	0	725.58	683.65	5.78	$ID_{random13}$	$D_{partition}$	3,897.02
26	0	678.97	643.14	5.28	$ID_{random24}$	$D_{partition}$	1,148.94
27	0	468.25	434.50	7.21	$ID_{random42}$	$D_{partition}$	3,224.60
28	0	407.95	362.70	11.09	$ID_{random44}$	$D_{partition}$	6,348.55
29	0	254.52	210.02	17.48	$ID_{random27}$	$D_{partition}$	2,894.33
30	0	280.40	229.95	17.99	$ID_{random44}$	$D_{partition}$	897.00
31	1	740.57	669.06	9.66	$ID_{random47}$	$D_{partition}$	4,021.57
32	0	89.12	65.41	26.61	$ID_{random28}$	$D_{partition}$	181.89

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	MCAD Runtime, T_{MCAD} [ms]
33	0	508.47	480.65	5.47	$ID_{random20}$	$D_{partition}$	1,578.35
34	0	897.48	841.03	6.29	$ID_{random6}$	$D_{partition}$	3,736.57
35	0	442.14	414.21	6.32	$ID_{random42}$	$D_{partition}$	1,232.39
36	0	459.51	435.33	5.26	$ID_{random34}$	$D_{partition}$	1,724.61
37	0	344.31	309.76	10.04	$ID_{random24}$	$D_{partition}$	674.43
38	0	190.12	153.04	19.50	$ID_{random1}$	$D_{partition}$	1,465.53
39	40	816.47	750.61	8.07	$ID_{random22}$	$D_{partition}$	11,877.46
40	0	427.66	396.05	7.39	$ID_{random28}$	$D_{partition}$	1,747.58
41	0	458.61	422.10	7.96	$ID_{random5}$	$D_{partition}$	5,094.38
42	0	792.61	736.91	7.03	$ID_{random16}$	$D_{partition}$	12,173.32
43	0	532.34	500.22	6.03	$ID_{random37}$	$D_{partition}$	4,298.13
44	0	513.45	481.79	6.17	$ID_{random33}$	$D_{partition}$	2,404.21
45	0	980.57	932.33	4.92	$ID_{random13}$	$D_{partition}$	3,832.46
46	0	610.61	573.02	6.16	$ID_{random35}$	$D_{partition}$	1,936.91
47	0	743.11	697.01	6.20	$ID_{random30}$	$D_{partition}$	6,357.58
48	0	581.60	547.72	5.83	$ID_{random25}$	$D_{partition}$	2,804.52
49	0	652.69	603.77	7.50	$ID_{random28}$	$D_{partition}$	15,231.59
50	0	618.55	586.13	5.24	$ID_{random23}$	$D_{partition}$	2,507.77
51	43	850.91	786.64	7.55	$ID_{random2}$	$D_{partition}$	7,703.13
52	0	664.31	624.80	5.95	$ID_{random14}$	$D_{partition}$	5,924.49
53	0	535.40	501.84	6.27	$ID_{random25}$	$D_{partition}$	3,172.59
54	0	1,091.04	1,047.92	3.95	$ID_{random30}$	$D_{partition}$	19,658.86
55	0	759.68	723.11	4.81	$ID_{random5}$	$D_{partition}$	1,706.02
56	25	603.16	505.13	16.25	$ID_{random15}$	$D_{partition}$	16,695.42
57	0	390.82	358.33	8.31	$ID_{random33}$	$D_{partition}$	1,548.49
58	1	525.84	500.47	4.83	$ID_{random49}$	$D_{partition}$	8,141.20
59	0	1,140.22	1,091.96	4.23	$ID_{random32}$	$D_{partition}$	4,273.15
60	25	421.38	341.40	18.98	$ID_{random50}$	$D_{partition}$	3,969.89
61	23	1,499.34	1,441.15	3.88	$ID_{random15}$	$D_{partition}$	58,631.56
62	0	624.24	587.97	5.81	$ID_{random44}$	$D_{partition}$	3,420.57
63	0	724.66	663.25	8.47	$ID_{random25}$	$D_{partition}$	34,196.68
64	2	951.47	884.74	7.01	$ID_{random23}$	$D_{partition}$	38,371.17
65	49	1,084.55	1,029.80	5.05	$ID_{random10}$	$D_{partition}$	1,682.36
66	0	147.84	134.11	9.28	$ID_{random22}$	$D_{partition}$	239.22

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	MCAD Runtime, T_{MCAD} [ms]
67	0	295.60	250.55	15.24	$ID_{random18}$	$D_{partition}$	866.81
68	0	296.73	252.33	14.96	$ID_{random20}$	$D_{partition}$	1,127.05
69	10	474.44	456.80	3.72	$ID_{random38}$	$D_{partition}$	1,297.00
70	0	430.79	400.11	7.12	$ID_{random31}$	$D_{partition}$	1,382.85
71	29	435.84	370.93	14.89	$ID_{random28}$	$D_{partition}$	16,419.79
72	14	632.10	571.09	9.65	$ID_{random33}$	$D_{partition}$	3,538.96
73	0	133.85	107.83	19.44	$ID_{random10}$	$D_{partition}$	244.89
74	48	896.56	847.23	5.50	$ID_{random27}$	$D_{partition}$	2,651.63
75	0	630.18	585.02	7.17	$ID_{random5}$	$D_{partition}$	9,926.70
76	0	576.61	534.90	7.23	$ID_{random15}$	$D_{partition}$	3,867.08
77	18	773.63	729.22	5.74	$ID_{random47}$	$D_{partition}$	16,503.21
78	22	556.89	493.27	11.42	$ID_{random5}$	$D_{partition}$	6,562.62
79	0	536.91	494.33	7.93	$ID_{random2}$	$D_{partition}$	2,452.77
80	33	704.85	651.20	7.61	$ID_{random13}$	$D_{partition}$	10,562.09
81	0	581.44	518.80	10.77	$ID_{random11}$	$D_{partition}$	24,037.93
82	0	620.99	582.67	6.17	$ID_{random38}$	$D_{partition}$	4,326.19
83	0	454.79	406.42	10.63	$ID_{random21}$	$D_{partition}$	2,052.97
84	47	822.33	723.32	12.04	$ID_{random14}$	$D_{partition}$	3,386.56
85	31	1,147.68	1,098.36	4.30	$ID_{random15}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	22,165.51
86	0	283.99	233.03	17.95	$ID_{random10}$	$D_{partition}$	2,328.14
87	8	911.44	849.04	6.85	$ID_{random32}$	$D_{partition}$	65,830.85
88	29	1,196.42	1,146.26	4.19	$ID_{random33}$	$D_{BP\text{-}HBF\text{-}Mem\text{-}Uncoarsen}$	41,117.02
89	34	805.24	732.02	9.09	$ID_{random2}$	$D_{partition}$	24,156.83
90	2	780.39	737.25	5.53	$ID_{random30}$	$D_{partition}$	4,244.22
91	3	885.32	825.14	6.80	$ID_{random7}$	$D_{partition}$	43,804.50
92	0	383.37	351.66	8.27	$ID_{random15}$	$D_{partition}$	1,405.46
93	0	468.97	435.71	7.09	$ID_{random16}$	$D_{partition}$	3,781.95
94	2	723.79	642.30	11.26	$ID_{random6}$	$D_{partition}$	73,347.29
95	0	197.53	166.37	15.77	$ID_{random43}$	$D_{partition}$	1,033.25
96	2	930.84	878.22	5.65	$ID_{random37}$	$D_{partition}$	25,825.29
97	1	1,025.26	981.84	4.24	$ID_{random29}$	$D_{partition}$	11,328.09
98	0	137.95	102.59	25.63	ID_{power}	$D_{partition}$	354.72
99	0	946.39	897.38	5.18	$ID_{random50}$	$D_{partition}$	18,077.53

Case	N_{bad}^{ID}	P_{MCAD} [W]	P_{bound} [W]	$\Delta_{\%}(P_{MCAD}, P_{bound})$ [%]	Solution Initial Deployment	Solution Deployment	MCAD Runtime, T_{MCAD} [ms]
100	0	516.02	483.55	6.29	$ID_{random4}$	$D_{partition}$	2,588.01
101	0	393.90	362.96	7.86	$ID_{random42}$	$D_{partition}$	928.90
102	0	675.59	631.07	6.59	$ID_{random37}$	$D_{partition}$	5,922.09
103	0	194.71	161.40	17.10	$ID_{random26}$	$D_{partition}$	969.23
104	8	420.75	372.24	11.53	$ID_{random20}$	$D_{partition}$	14,826.76

Table 74 – Results for validating the tuned MCAD algorithm (Part 2)

Case	Processing Time, R_{PT} [ms]	Network Delay, R_{net_delay} [ms]	Response Time Algorithm, R_{MCAD} [ms]	Response Time from the LQNS, R_{LQNS} [ms]	$\Delta_{\%}(R_{MCAD}, R_{LQNS})$ [%]	Time taken by the LQNS to solve the LQN model, T_{LQNS} [s]	Number of iterations taken by the LQNS to solve the LQN model, N_{LQNS}^{iter}
1	62.48	1,964.75	2,027.23	1979.31	2.36	371.64	9
2	62.89	2,240.11	2,303.00	2254.23	2.12	11.52	6
3	51.32	1,260.08	1,311.40	1272.02	3.00	42.26	8
4	22.12	819.31	841.43	824.378	2.03	10.49	9
5	76.45	2,589.01	2,665.46	N/A	N/A	N/A	N/A
6	56.97	1,101.74	1,158.71	1114.88	3.78	3.85	7
7	59.5	1,626.55	1,686.05	1639.84	2.74	61.17	7
8	28.68	802.34	831.02	809.094	2.64	4.81	14
9	17.97	355.02	372.99	359.333	3.66	1.70	9
10	29.11	507.97	537.08	514.902	4.13	208.18	9
11	157.61	2,992.24	3,149.86	N/A	N/A	N/A	N/A
12	28.42	923.03	951.45	929.661	2.29	6.59	7
13	36.87	1,581.69	1,618.56	1590.23	1.75	19.35	50
14	130.07	1,716.15	1,846.22	N/A	N/A	N/A	N/A
15	18.57	942.72	961.29	1071.04	-11.42	327.30	8
16	72.87	2,278.12	2,350.99	2294.67	2.40	10.17	9
17	88.29	1,425.49	1,513.78	1936.26	-27.91	691.80	9
18	23.18	272.27	295.46	N/A	N/A	N/A	N/A
19	8.15	301.44	309.59	303.261	2.04	4.87	7
20	23.04	548.28	571.31	553.627	3.10	54.89	9
21	230.56	5,796.70	6,027.26	6896.51	-14.42	12.82	11
22	353.98	6,666.99	7,020.97	7024.35	-0.05	481.99	7
23	433.09	3,771.30	4,204.39	5490.29	-30.58	306.61	12
24	79.31	2,642.31	2,721.62	2662.01	2.19	0.06	9
25	503.77	6,948.00	7,451.77	8767.32	-17.65	1,177.67	11
26	405.75	6,892.30	7,298.05	7403.9	-1.45	21.80	10
27	538.84	6,901.98	7,440.82	7599.49	-2.13	338.85	11
28	772.63	6,001.87	6,774.50	6924.85	-2.22	174.67	11

Case	Processing Time, R_{PT} [ms]	Network Delay, R_{net_delay} [ms]	Response Time Algorithm, R_{MCAD} [ms]	Response Time from the LQNS, R_{LQNS} [ms]	$\Delta_{\%}(R_{MCAD}, R_{LQNS})$ [%]	Time taken by the LQNS to solve the LQN model, T_{LQNS} [s]	Number of iterations taken by the LQNS to solve the LQN model, N_{LQNS}^{iter}
29	481.63	6,512.51	6,994.14	9213.87	-31.74	104.89	9
30	335.85	6,903.61	7,239.47	7063.19	2.43	2.04	8
31	675.78	6,769.50	7,445.29	6923.12	7.01	76.99	10
32	137.04	2,838.05	2,975.10	3505.09	-17.81	0.10	8
33	184.86	3,457.16	3,642.01	3911.18	-7.39	106.57	11
34	411.24	6,931.99	7,343.23	N/A	N/A	N/A	N/A
35	196.49	5,984.70	6,181.19	6635.11	-7.34	17.00	9
36	257.94	6,333.31	6,591.25	8118.67	-23.17	42.54	11
37	329.22	4,671.56	5,000.78	5230.84	-4.60	1.80	11
38	193.57	5,278.31	5,471.88	5749.74	-5.08	2.00	8
39	615.79	6,881.36	7,497.16	N/A	N/A	N/A	N/A
40	248.84	6,653.69	6,902.53	8099.2	-17.34	40.93	10
41	596.59	9,149.39	9,745.98	N/A	N/A	N/A	N/A
42	743.01	9,247.13	9,990.15	10927.5	-9.38	975.26	12
43	416.84	5,416.85	5,833.69	6893.06	-18.16	1,367.85	11
44	415.14	7,371.83	7,786.97	7591.72	2.51	716.56	12
45	1,469.97	16,848.71	18,318.68	N/A	N/A	N/A	N/A
46	1,088.88	12,618.32	13,707.20	15143	-10.47	71.54	14
47	524.6	6,668.99	7,193.58	N/A	N/A	N/A	N/A
48	383.69	4,467.71	4,851.40	7134.12	-47.05	301.74	11
49	647.71	9,350.31	9,998.03	N/A	N/A	N/A	N/A
50	374.3	7,803.95	8,178.25	9083.37	-11.07	406.97	11
51	1,110.24	8,888.22	9,998.46	N/A	N/A	N/A	N/A
52	684.25	8,814.57	9,498.83	N/A	N/A	N/A	N/A
53	1,131.29	16,854.95	17,986.24	18972.2	-5.48	234.15	12
54	1,951.92	17,836.66	19,788.57	N/A	N/A	N/A	N/A
55	1,176.54	10,440.17	11,616.71	14857	-27.89	100.68	13
56	4,904.25	9,945.11	14,849.36	13827.9	6.88	1,431.98	29
57	2,182.00	12,210.46	14,392.46	16525.5	-14.82	10.69	19

Case	Processing Time, R_{PT} [ms]	Network Delay, R_{net_delay} [ms]	Response Time Algorithm, R_{MCAD} [ms]	Response Time from the LQNS, R_{LQNS} [ms]	$\Delta_{\%}(R_{MCAD}, R_{LQNS})$ [%]	Time taken by the LQNS to solve the LQN model, T_{LQNS} [s]	Number of iterations taken by the LQNS to solve the LQN model, N_{LQNS}^{iter}
58	3,890.06	10,997.44	14,887.50	14833.4	0.36	146.47	17
59	981.64	9,585.08	10,566.72	9908.06	6.23	1,730.01	16
60	5,447.52	9,338.18	14,785.70	11700.2	20.87	12.84	23
61	2,256.71	12,400.44	14,657.16	N/A	N/A	N/A	N/A
62	749.29	14,192.51	14,941.80	14545.9	2.65	166.90	19
63	1,225.73	13,772.72	14,998.45	N/A	N/A	N/A	N/A
64	1,590.72	13,363.77	14,954.49	N/A	N/A	N/A	N/A
65	11,004.15	3,870.23	14,874.38	18916.3	-27.17	323.09	50
66	4,856.73	2,511.26	7,367.99	4090.16	44.49	0.02	23
67	21,279.01	77,826.22	99,105.23	83705	15.54	0.88	17
68	414.82	4,353.94	4,768.77	4746.43	0.47	4.17	14
69	2,195.23	12,322.15	14,517.37	12901.4	11.13	2.68	20
70	471.02	7,534.51	8,005.53	9017.29	-12.64	28.17	21
71	4,640.87	10,083.83	14,724.70	12291.3	16.53	756.03	25
72	4,000.55	10,865.57	14,866.12	12103.2	18.59	13.59	27
73	450.22	882.62	1,332.85	1014.98	23.85	0.02	15
74	42,070.18	157,306.20	199,376.38	167830	15.82	1,722.43	20
75	1,565.54	8,368.04	9,933.58	10158.7	-2.27	1,544.24	18
76	1,250.32	8,704.93	9,955.25	10620.4	-6.68	17.63	21
77	7,385.94	22,369.56	29,755.50	37067.1	-24.57	245.92	28
78	4,332.41	25,502.42	29,834.84	40977.2	-37.35	155.27	23
79	1,086.81	8,824.38	9,911.19	9635.99	2.78	11.97	19
80	6,832.50	23,027.40	29,859.90	25499.3	14.60	478.72	26
81	23,918.69	175,950.40	199,869.08	235609	-17.88	1,726.54	20
82	5,415.13	23,913.93	29,329.05	29951.2	-2.12	194.51	22
83	30,143.79	319,774.40	349,918.20	327768	6.33	51.23	24
84	58,586.96	287,490.62	346,077.58	N/A	N/A	N/A	N/A
85	65,657.46	133,741.32	199,398.77	N/A	N/A	N/A	N/A
86	11,532.69	179,970.04	191,502.74	205135	-7.12	27.06	23

Case	Processing Time, R_{PT} [ms]	Network Delay, R_{net_delay} [ms]	Response Time Algorithm, R_{MCAD} [ms]	Response Time from the LQNS, R_{LQNS} [ms]	$\Delta_{\%}(R_{MCAD}, R_{LQNS})$ [%]	Time taken by the LQNS to solve the LQN model, T_{LQNS} [s]	Number of iterations taken by the LQNS to solve the LQN model, N_{LQNS}^{iter}
87	9,261.87	20,581.81	29,843.68	N/A	N/A	N/A	N/A
88	36,914.87	162,844.73	199,759.60	176532	11.63	1,372.73	23
89	2,521.48	7,410.28	9,931.75	8532.73	14.09	506.47	21
90	5,145.58	24,653.77	29,799.35	29612	0.63	39.40	20
91	14,666.18	84,537.01	99,203.19	N/A	N/A	N/A	N/A
92	5,213.98	81,875.22	87,089.21	92701	-6.44	4.88	18
93	8,160.54	90,965.74	99,126.28	96388.2	2.76	78.33	20
94	14,598.48	85,384.46	99,982.94	121532	-21.55	1,413.87	21
95	1,420.17	23,138.35	24,558.52	26676.6	-8.62	0.97	21
96	18,965.89	130,518.97	149,484.86	N/A	N/A	N/A	N/A
97	11,401.76	86,303.93	97,705.69	92014.1	5.83	573.54	20
98	4,691.08	18,718.29	23,409.37	24413.7	-4.29	0.06	24
99	11,730.61	87,442.94	99,173.55	N/A	N/A	N/A	N/A
100	7,150.78	75,130.87	82,281.64	83585.4	-1.58	33.11	20
101	3,836.17	57,575.29	61,411.46	59147.2	3.69	3.16	24
102	7,716.85	90,866.43	98,583.28	93529	5.13	1,237.34	23
103	5,172.88	35,384.16	40,557.04	53845	-32.76	0.29	21
104	37,141.53	112,515.88	149,657.40	137471	8.14	209.24	38