

**A NETWORK SCIENCE BASED APPROACH FOR  
OPTIMAL MICROSERVICE GOVERNANCE**

2020-021

Project Proposal Report

L.S. Jayasinghe

B.Sc. (Hons) in Information Technology Specializing in Software  
Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology  
Sri Lanka

February 2020

**A NETWORK SCIENCE BASED APPROACH FOR  
OPTIMAL MICROSERVICE GOVERNANCE**

2020-021

# Individual Project Proposal Report

B.Sc. (Hons) in Information Technology Specializing in Software  
Engineering

Department of Software Engineering


Sri Lanka Institute of Information Technology  
Sri Lanka

February 2020

## Declaration

I declare that this is my own work and this proposal does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any other university or institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Signature:

IT 17012966	L.S Jayasinghe	
-------------	----------------	---

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

Signature of the Supervisor:  .....

Date: 23/02/2020 .....

## **Abstract**

This research is regarding optimal deployment algorithm for microservices automated deployment. Algorithm will ensuring that application has optimal overall performance and availability. It maximizes performance and availability using the multi-objective algorithm.

there are several input metrics are used to generate an optimal deployment plan. A load prediction map is used to identify the strongest dependencies of the microservice application and resiliency level of the application used to identify critical microservices. also, system concern about individual nodes resource power and individual microservices resource consumption.

Using previously mentioned input metrics, the system minimizes the overall network latency.it causes better performance. Also, it maintains the application overall availability. generating randomize deployment plans, the system chooses the optimal deployment plan. finally, that plan will perform the automated deployment.

**Keywords:** Auto-scaling, Docker, Kubernetes, Machine Learning, Microservices

## TABLE OF CONTENTS

<b>Declaration .....</b>	<b>i</b>
<b>Abstract .....</b>	<b>ii</b>
<b>List of Figures .....</b>	<b>v</b>
<b>List of Tables .....</b>	<b>vi</b>
<b>List of Abbreviations .....</b>	<b>vii</b>
<b>1.0 Introduction .....</b>	<b>1</b>
<b>1.1 Background and Literature .....</b>	<b>4</b>
<b>1.3 Research Problem .....</b>	<b>8</b>
<b>2.0 Objectives .....</b>	<b>9</b>
<b>2.1 Main Objective .....</b>	<b>9</b>
<b>2.2 Specific Objectives .....</b>	<b>9</b>
<b>3.0 Methodology .....</b>	<b>10</b>
<b>3.1 Requirement Gathering .....</b>	<b>10</b>
<b>3.1.1 Past Research Analysis .....</b>	<b>10</b>
<b>3.1.2 Identifying Existing Systems .....</b>	<b>10</b>
<b>3.2 Feasibility Study .....</b>	<b>11</b>
<b>3.2.1 Technical Feasibility .....</b>	<b>11</b>
<b>3.2.1.5 Optimization algorithm. ....</b>	<b>12</b>
<b>3.2.1.6 cloud network and Kubernetes network. ....</b>	<b>12</b>
<b>3.2.2 Schedule Feasibility .....</b>	<b>13</b>
<b>3.2.3 Economic Feasibility .....</b>	<b>13</b>
<b>3.3 Requirement Analysis .....</b>	<b>13</b>
<b>3.4 System Analysis .....</b>	<b>14</b>
<b>3.4.1 Software Solution .....</b>	<b>14</b>
<b>3.4.1.1 Input Metrics .....</b>	<b>14</b>
<b>3.5 System Development and Implementation .....</b>	<b>16</b>
<b>3.5.1 Building the Final Optimization Algorithm .....</b>	<b>17</b>
<b>3.6 Project Requirements .....</b>	<b>18</b>
<b>3.6.1 Functional Requirements .....</b>	<b>18</b>
<b>3.6.2 Non-Functional Requirements .....</b>	<b>18</b>
<b>3.7 Testing .....</b>	<b>19</b>

<b>3.8 Time Line.....</b>	<b>21</b>
<b>4.0 Personnel and Facilities.....</b>	<b>22</b>
<b>5.0 Commercialization .....</b>	<b>23</b>
<b>6.0 Budget.....</b>	<b>24</b>
<b>7.0 Summary .....</b>	<b>25</b>
<b>8.0 References.....</b>	<b>26</b>
<b>9.0 Appendices.....</b>	<b>27</b>

## List of Figures

Figure 1.1 Network Latency Between Multiple Region .....	3
Figure 1.1.1 Travelling Salesman Problem .....	4
Figure 1.1.2 Genetic Algorithm Parts .....	5
Figure 1.1.3 Process Of Genetic Algorithm. ....	6
Figure 3.2 Proposed Methodology .....	16
Figure 3.3 Gantt Chart .....	21

## List of Tables

Table 4.1 Personnel and Resources .....	21
Table 6.1 Budget.....	23



## List of Abbreviations

Abbreviation	Description
AKS	Azure Kubernetes Services
APM	Application Performance Monitoring
AWS	Amazon Web Service
IT	Information Technology
SOA	Service-Oriented Architecture
TOSCA	Topology and Orchestration Specification for Cloud Applications
VM	Virtual Machine
IBM	International Business Machines Corporation

## 1.0 Introduction

In 2020, most of the large application companies move to microservice architecture. Such as Facebook, Netflix, and Amazon etc. These companies expect high availability, maintainability, version control and performance from microservice architecture. There are lot of configuration issues related to microservices. In 2013 docker is introduced[1]. It is the solution that microservice is put into the single docker containers.

Then it is the solution for most of the configuration's issues. Now with a docker container, the application can be deployed anywhere without any hesitation.

Problems are not over. Generally, microservice related applications used more than 100 microservices. Even Netflix used 700 microservices[2]. Now problem is the management of these microservices. To solve this problem, container-orchestration tools introduced. Kubernetes and Docker swarm are a few examples of container-orchestration tools[3].

Availability is the most required none functional requirement in the microservices model. The low availability cause of most of the system fails. To increase availability, we can use multiple instances per microservices.it is called replication in the microservice model. now one instance failed, there is another instance for the job done. If all the instances are deployed on single nodes and that nodes are down, no instances are available for executes task[4].

So multiple instances are deployed over the multiple nodes is very important to gain more availability. Also if multiple nodes are located in multiple data centers, It gains more availability.

The Microservice model is a great solution for gaining availability in the application. But it dramatically reduces the whole application performance.

IBM Researchers said,

“We observed a significant overhead due to the microservice architecture; the performance of the microservice model can be 79.1% lower than the monolithic model on the same hardware configuration. The microservice model spent much more time in runtime libraries to process one client request than the monolithic model by 4.22x on a Node.js application server and by 2.69x on a Java EE application server.”[5]

By analyzing the above statement. We can identify the performance as a main issue in the microservice architecture. In considering the above scenario, the Microservice model 79.1% low performance than the monolithic model.

If we consider the performance of the web application, we can measure performance using the response time. If the average response time is lower across the whole application, it can be identified as a good performance application. Computational time, bandwidth and network latency directly affect response time. However a bigger portion of the time reserve for the network latency. This metric is calculated by using a round trip time[7].

It is measured in milliseconds. Even if the network has a high bandwidth rate, high network latency time takes a long time to get the response back. This effect can be experienced in the multiplayer gaming network. Because lower ping (RTT) players play well than the other players.

in the microservice model, microservices intercommunicate with the rest API calls. Also, these services are deployed over multiple nodes.so even send one request, it goes through multiple nodes.

In the cluster network, nodes are interconnected with network links, when the request goes through these links, network latencies added for each link. Also, if the nodes are located very far, it increases the network latency. So that it reduces the overall performance of the application[7].

	Australia Central	Australia Central2	Australia East	AustraliaSoutheast	BrazilSouth	Canada Central	Canada East	Central India	Central US	East Asia	East US	East US2	France Central	FranceSouth	Germany North	Germany West Central	Japan East	Japan West	Korea Central	Korea South	North Central US	North Europe	Norway East	Norway West	South Central US	South East Asia	South India	SouthAfrica West	SouthAfricaNorth	Switzerland North	Switzerland West	UAE Central	UAE North	UK South	UK West	West Central US	West Europe	West India	West US	West US2
Australia Central	1	6	18	312	204	214	148	184	121	205	200	290	304	302	296	131	136	150	146	192	276	312	304	176	94	126	392	388	302	306	186	186	285	288	170	292	146	142	162	
Australia Central2	1	6	18	312	206	214	146	186	120	206	202	291	304	302	296	130	134	148	146	192	278	312	306	176	92	124	392	388	302	306	186	186	286	288	172	292	144	144	164	
Australia East	6	6	12	306	200	210	142	182	116	200	196	286	300	296	292	126	130	144	142	188	272	308	300	170	88	120	388	382	297	300	182	180	280	282	166	288	140	140	158	
AustraliaSoutheast	18	18	12	318	210	220	140	192	118	210	206	296	310	308	302	132	136	146	140	198	282	318	310	182	86	118	386	382	308	312	178	178	290	294	178	297	138	150	170	
Brazil South	312	312	306	318	132	140	302	144	322	118	114	186	200	196	192	260	268	301	306	138	172	208	200	138	326	316	328	342	198	200	294	300	180	184	159	188	302	170	180	
Canada Central	204	206	200	210	132	12	208	20	198	25	30	92	106	104	98	152	160	178	182	15	80	114	108	44	216	222	238	248	104	108	202	206	86	90	36	94	208	64	56	
Canada East	214	214	210	220	140	12	218	30	208	35	38	102	116	114	108	162	170	186	192	24	88	124	116	52	226	232	246	256	114	118	210	214	96	98	44	104	216	72	66	
Central India	148	146	142	140	302	208	218	222	88	196	198	116	104	128	120	122	122	116	110	216	132	146	140	224	54	24	264	262	114	110	30	29	124	126	236	126	4	222	214	
Central US	184	186	182	192	144	20	30	222	176	27	34	106	112	118	112	132	140	157	162	32	94	128	120	22	196	228	246	260	120	114	216	220	100	104	14	110	222	42	36	
East Asia	121	120	116	118	322	198	208	88	176	200	208	184	172	196	188	50	50	52	56	184	198	214	206	182	34	66	334	328	182	178	126	126	191	194	162	194	86	149	142	
East US	205	206	200	210	118	25	35	196	27	200	6	80	90	90	86	168	168	181	186	18	66	102	94	32	220	209	220	234	92	88	188	192	74	76	47	82	194	64	66	
East US2	200	202	196	206	114	30	38	198	34	208	6	80	90	94	90	164	164	176	182	24	70	106	98	28	226	212	222	238	93	88	190	194	78	80	48	85	196	59	71	
France Central	290	291	286	296	186	92	102	116	106	184	80	80	12	20	218	216	212	206	98	16	32	24	106	151	129	152	166	14	10	108	112	8	8	120	10	116	138	142		
FranceSouth	304	304	300	310	200	106	116	104	112	172	90	90	12	26	17	204	202	200	194	106	26	44	36	116	138	116	164	158	10	8	96	100	17	18	132	20	102	148	152	
Germany North	302	302	296	308	196	104	114	128	118	196	90	94	20	26	10	229	228	224	218	110	26	20	26	121	162	142	164	178	16	19	120	124	20	22	132	10	126	160	154	
Germany West Central	296	296	292	302	192	98	108	120	112	188	86	90	10	17	10	220	218	216	210	106	20	28	20	116	154	132	160	174	6	10	112	116	14	18	126	8	118	154	148	
Japan East	131	130	126	132	260	152	162	122	132	50	168	164	218	204	229	220	8	28	34	139	232	247	240	138	68	100	374	362	214	211	161	160	224	228	118	226	120	104	96	
Japan West	136	134	130	136	268	160	170	122	140	50	168	164	216	202	228	218	8	36	42	147	232	246	238	138	76	100	366	360	214	210	160	160	224	226	126	225	120	106	104	
Korea Central	150	148	144	146	301	178	186	116	157	52	181	176	212	200	224	216	28	36	6	164	228	242	234	150	62	94	362	358	210	206	156	154	220	222	142	222	114	129	122	
Korea South	146	146	142	140	306	182	192	110	162	56	186	182	206	194	218	210	34	42	6	170	221	236	228	156	56	88	356	352	204	200	150	148	212	215	148	216	108	134	126	
North Central US	192	192	188	198	138	15	24	216	12	184	18	24	98	106	110	106	139	147	164	170	84	122	114	34	202	228	244	256	110	106	209	212	92	96	26	101	214	54	46	
North Europe	276	278	272	282	172	80	88	132	94	198	66	70	16	26	26	20	232	232	228	221	84	36	30	96	166	145	150	172	26	24	124	130	10	12	108	16	130	128	130	
Norway East	312	312	308	318	208	114	124	146	128	214	102	106	32	44	20	28	247	246	242	236	122	36	8	132	180	158	174	186	32	36	138	142	29	30	142	22	144	170	164	
Norway West	304	306	300	310	200	108	116	140	120	206	94	98	24	36	26	20	240	238	234	228	114	30	8	124	174	152	168	180	26	30	130	135	22	24	136	15	138	164	156	
South Central US	176	176	170	182	138	44	52	224	12	182	32	28	106	116	121	116	138	138	150	156	34	96	132	124	200	222	252	264	120	116	218	220	104	106	22	112	222	34	44	
South East Asia	94	92	88	86	326	216	226	54	196	34	220	226	151	138	162	154	68	76	62	56	202	166	180	174	200	34	300	296	148	144	94	94	158	162	182	160	52	168	160	
South India	126	124	120	118	316	222	232	24	228	66	209	212	129	116	142	132	100	100	94	88	228	145	158	152	222	34	279	274	126	124	48	48	136	140	214	138	22	200	192	
SouthAfrica West	392	392	388	386	328	238	246	264	246	334	220	222	152	164	164	160	374	366	362	356	244	150	174	168	252	300	279	18	164	160	258	262	148	150	264	151	265	292	284	
SouthAfricaNorth	388	388	382	382	342	248	256	262	260	328	234	238	166	158	178	174	362	360	358	352	256	172	186	180	264	296	274	18	170	166	254	258	162	164	276	170	260	304	297	
Switzerland North	302	302	297	308	198	104	114	114	120	182	92	93	14	10	16	6	214	214	210	204	110	26	32	26	120	148	126	164	170	4	106	110	20	22	134	12	114	150	156	
Switzerland West	306	306	300	312	200	108	118	110	114	178	88	88	10	8	19	10	211	210	206	200	106	24	36	30	116	144	124	160	166	4	102	106	16	18	130	16	110	148	153	
UAE Central	186	186	182	178	294	202	210	30	216	126	188	190	108	96	120	112	161	160	156	150	209	124	138	130	118	94	48	258	254	106	102	4	116	118	230	118	28	258	252	
UAE North	186	186	180	178	300	206	214	29	220	126	192	194	112	100	124	116	160	160	154	148	212	130	142	135	220	94	48	262	258	110	106	4	120	122	234	122	28	262	256	
UK South	285	286	280	290	180	86	96	124	100	191	74	78	8	17	20	14	224	224	220	212	92	10	29	22	104	158	136	148	162	20	16	116	120	3	114	10	122	136	135	
UK West	288	288	282	294	184	90	98	126	104	194	76	80	8	18	22	18	228	226	222	215	96	12	30	24	106	162	140	150	164	22	18	118	122	9	116	12	124	138	138	
West Central US	170	172	166	178	159	36	44	236	14	162	47	48	120	132	132	126	118	126	142	148	26	108	142	136	22	182	214	264	276	134	130	230	234	114	116	124	236	22		
West Europe	292	292	288	297	188	94	104	126	110	194	82	85	10	20	10	8	226	225	222	216	101	16	22	15	112	160	138	151	170	12	16	118	122	10	12	124	126	144	146	
West India	146	144	140	138	302	208	216	4	222	86	194	196	116	102	126	118	120	114	108	214	130	144	138	222	52	22	265	260	114	110	28	28	122	124	236	126	220	212		
West US	142	144	140	150	170	64	72	222	42	149	64	59	138																											

## 1.1 Background and Literature

There are not any tools that come up with the algorithm for optimal deployment strategy. But there are a lot of usages of optimization algorithms.

One of the Most popular optimization problems is the traveling salesman problem. The problem is to generate the shortest pathing route and visits to each city per once a time. also, the path should be covering all the given cities and return to the origin city. It is called the NP-hard problem[9].

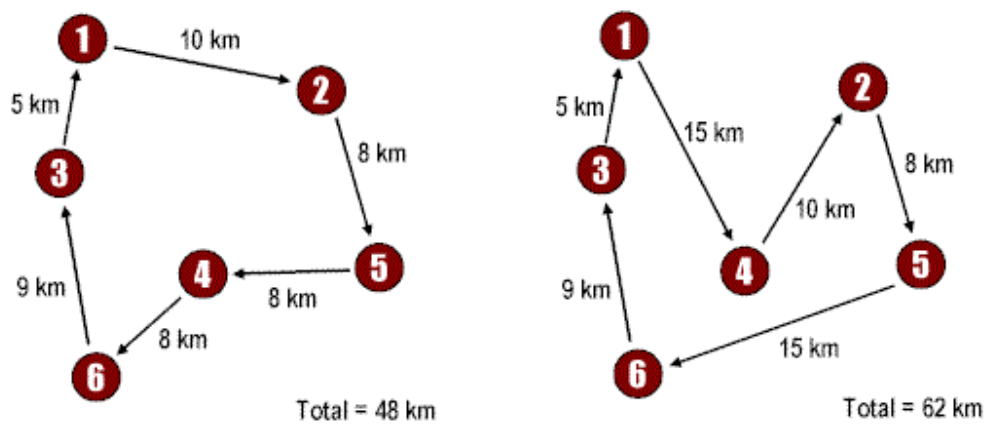


Figure 1.1.1 Travelling Salesman Problem Graph

To solve TSP, programmers can use exact algorithms. These algorithm is trying all combination of pathing solutions (brute-force search). Among the solution, the algorithm selects the cheapest one. It is the shortest path among all solutions. Always the Exact algorithm gives the best solution. But problems will come when the number of combinations is increased. This consumes substantial computational power and time.

Heuristic and approximation algorithms used to avoid this previous issue.it always generates a better solution but it is not the best solution. Compare with the exact algorithm, heuristic algorithm consumes very less time[10].

### Heuristic Algorithms:

- swarm intelligence
- tabu search
- simulated annealing
- genetic algorithm

In this research, the genetic algorithm will be used to solve our problems. The genetic algorithm was introduced in 1960, based on the theory of evolution. Optimal algorithm problems are solved by using biologically inspired operations. Such as mutation, crossover, and selection[11][12].

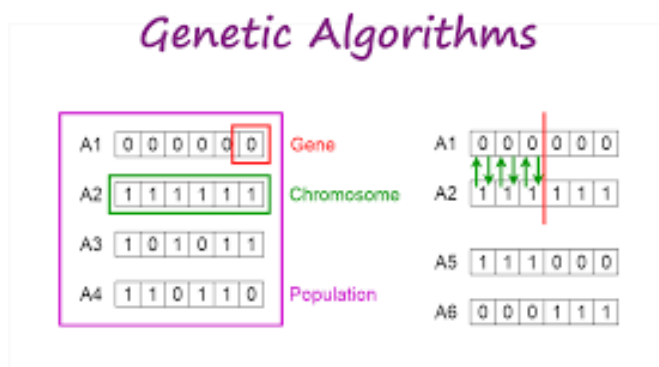


Figure 1.1.2 Genetic Algorithm Parts

### Part of the genetic algorithm

- Chromosome: a single solution.
- Genes: one element of a single solution.
- population: the process is started by the beginning of the initial population. fittest chromosomes are selected to the next generation by calculating the fitness score.
- Fitness function: for selecting the next generation, genetic algorithm use fitness function.it has a score calculation method.

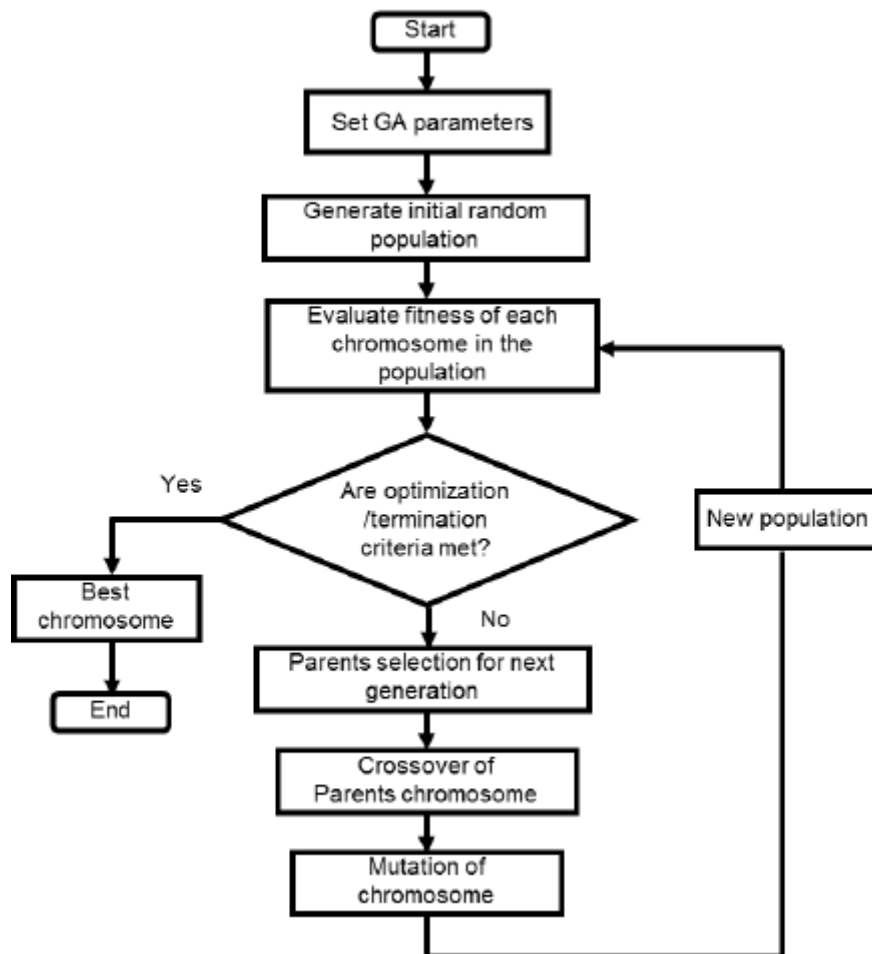


Figure 1.1.3 Process Of Genetic Algorithm

Operations of the genetic algorithm:

- Selection: this is done by the fitness function.
- Crossover: combine two chromosomes.
- Mutation: modify single genes or multiple genes

## **1.2 Research Gap**

There is one related research found on the web. It is an optimal and automated deployment for microservices. That research describes how the microservices deploy over nodes and automated deployment. To implement their model system, they used a dependency map, each microservice resource consumption, each node resource power and cost[13].

In the research paper, they do not mention that the system model can increase performance or availability. But they mention that minimizing the total cost is their optimization problem.

In our research, we mainly focus on microservices performance and availability. The system will use predicted dependency map, node resource power, each microservice resource consumption, resiliency levels and node latency map as input metrics.

The system will guarantee to generate better optimal deployment than existed deployment plan. It should be better performance and the optimal availability than the previous plan.



### 1.3 Research Problem

All the application system has pre-defined non- functional requirements. Such as availability, adaptability, durability, interoperability, reliability, etc. In this research, we are mainly focused on performance and reliability.

When microservices are deployed in the cloud cluster, K8s is used for our primary tools. Using this tool, we can also perform the automated deployment. But it is not intelligent. It only has done which are scheduled by us. Also when new microservice comes, it is difficult to maintain the whole system.

Nodes are located in multiple places in the cloud. Some scenarios, it is located in another country. It is called a region. In the same region, there are multiple data centers. These distances create network latency within a cluster. Even in the same data center, there is some network latency[8].

If whole microservice applications deploy in one virtual machine. we can avoid from previously mentioned network latency.so that it also increases performance. But the problem is that the solution reduces availability.

Before microservices are deployed, we have no idea about current microservices resource consumption and cluster resource power (including all the nodes). After application is deployed, if resource power is not enough to execute all the requested tasks, the whole application will be crash. Also, if resource power is larger than the required resource power, it wastes a lot of money.

Our main target is to increased overall performance without losing availability. Also, each microservices should be well fit node resource power.

## **2.0 Objectives**

### **2.1 Main Objective**

To identify key factors that lead to performance reduction in microservice deployments and come up with an optimal deployment strategy .

### **2.2 Specific Objectives**

The following are the sub-objectives of conducting this research.

- To increase the efficiency of microservices deployments by applying the metrics used in network analysis, such as centrality and resilience measures, and link predictions on identified dependency measurements.
- To develop a business intelligence dashboard to evaluate performance and monitor microservice deployments.
- To identify key factors that lead to performance reduction in microservice deployments and come up with an optimal deployment strategy.

## **3.0 Methodology**

### **3.1 Requirement Gathering**

Requirement gathering was mainly performed through performing an extensive analysis of past research conducted throughout recent years, identification and analysis of the existing systems, as well as reading through a variety of online resources.

#### **3.1.1 Past Research Analysis**

The past research analysis process was mainly performed through reading and analyzing a wide array of research publications published through recent years. Key topics of interest included microservice deployment optimization, microservice performance engineering, microservice governance, centrality evaluation, load prediction and forecasting, resource prediction and optimization, resiliency analysis, and microservice monitoring. During the research analysis process, the primary focus was given in the identification of the methodology used, tools used, experiments conducted, as well as the overall findings of the research with respect to performance optimization in microservices.

#### **3.1.2 Identifying Existing Systems**

A thorough analysis was conducted on a variety of existing APM tools as well as other similar systems, that were available to use with the Kubernetes platform. This process was mainly done by visiting the various online sources and analyzing the available documentation and videos published. During this process, the primary focus was given in identifying the key features and drawbacks that were present in the tools analyzed.

Even if there is no existing system directly related to this research. There are automated deployment tools and related researches. That would be useful for testing the final product.

## **3.2 Feasibility Study**

### **3.2.1 Technical Feasibility**

#### **3.2.1.1 Knowledge on Kubernetes**

In order to develop the proposed optimization model, all members are required to have basic knowledge of Kubernetes and its relevant components. Members should be able to perform basic configuration and should have sufficient knowledge and practical experience on how to deploy microservices through Kubernetes.

#### **3.2.1.2 Knowledge on APM tools**

In order to develop the proposed optimization model, all members are required to have quite an in-depth understanding of the existing APM tools and the features and drawbacks present. The members should also have sufficient knowledge of configuring and APM tools selected for this research as well as knowledge on how to integrate the selected APM tools with the optimization model.

#### **3.2.1.3 Knowledge on Resiliency Evaluation and Chaos Engineering**

For the resiliency evaluation system to be developed, a thorough knowledge of “Chaos Engineering” and knowledge related Chaos Engineering tools are required. Furthermore, members should have sufficient knowledge of configuring and integrating the selected Chaos Engineering tool with the optimization model.

#### **3.2.1.4 Machine Learning Knowledge**

In order to develop the proposed optimization model, all members are required to have quite basic knowledge of machine learning basics as well as time series analysis. Members should be aware of the time series prediction models as well as knowledge on how to integrate the relevant models with machine learning and develop basic algorithms. Furthermore, members should also have a basic understanding of the Python programming language and related Python machine learning and time series libraries.

#### **3.2.1.5 Optimization algorithm.**

Optimization algorithms use a wide variety of fields. It is an applied mathematics field that part of data science. These optimization algorithms reach the optimal solution by maximizing or minimizing variables. Different types of optimal algorithms have different accuracy levels and performance levels. It is very important to select a better algorithm for producing a better result.

#### **3.2.1.6 cloud network and Kubernetes network.**

Cloud network consists of WAN, MAN, LAN and virtual networks. Also Kubernetes network includes node network and pod network. We have awareness about how to communicate between microservice using previously mentioned networks.

### **3.2.2 Schedule Feasibility**

The proposed project should be able to be implemented within the scheduled time period of about five months, with about two months allocated for research, requirement gathering, and analysis. Finally, the proposed project should be completed within the end of 7 months, including sufficient testing.

### **3.2.3 Economic Feasibility**

The cost of the proposed project should be as minimal as possible in order for it to be included and accepted in the existing APM tool market in Kubernetes. This is mainly due to the fact that most APM tools and solutions offered currently with respect to Kubernetes are often opensource.

## **3.3 Requirement Analysis**

During the requirement analysis phase, key information obtained during the requirement gathering phase is analyzed. Analyzing the gathered information will prove to be of most importance to the research process, since key information regarding the potential challenges that may be faced, the potential complexity of tasks involved, as well as other key information regarding the tools used by other research teams will be easily identifiable.

Also, since the research carried out a software-based approach, by performing requirement analysis, key information regarding the schedule, technical and economic feasibility was realized and helped in aligning research goals such that the research carried out does not exceed the technical skills of the research members while maintaining the expected deadlines.

In the research paper analysis, the primary focus will be given to the analysis of the methodology and tools used, as well as the outcomes of the research conducted. This helps in improving the decision-making process in the current research by providing credible evidence that will help in deciding upon the direction in which the current research should progress by highlighting the research gaps.

Furthermore, analyzing the online resources regarding the available tools will help in the identification of the existing tools that possess similar features to what the current research aims at implementing and help in identifying the research gap by comparing the existing feature with those that are proposed. Also, by analyzing tools that could prove to be of use in the implementation of the current research, a clear idea regarding the features they possess, and how they could be integrated into the current research could be identified.

### **3.4 System Analysis**

#### **3.4.1 Software Solution**

##### **3.4.1.1 Input Metrics**

There are several input metrics go through optimal deployment algorithm. Such as

- Predicted Dependency Map

This is the predicted dependency map that describes the average number of requests between pairs of microservices in a certain time period.

- Node Latency Map

This describe network latency between nodes.

- Resiliency Level

This describe resiliency level of each node.

- System Runtime Information
  - i. Node resource power
  - ii. Microservice resource consumption

### **3.4.1.3 Analysis and Rearrangement**

The first system obtains pairs of microservice dependency levels from the predicted dependency map. these pairs are rearranged according to strong dependency. as an example, most communicated a couple of microservices have the strongest dependency.

After that, analyze the resiliency level of microservices. Using a resiliency methods system can identify the most critical microservices in the System. This metric can be used to increase availability.

Next, calculate the quickest path between node by using node latency map. near located nodes have low network latency than far located nodes. If most communicate couple of microservice located in near nodes, whole application makes more score

These metrics are rearranged according to the importance level. Most important metrics more effect on the fitness score function.

### **3.4.1.4 Genetic Algorithm for Optimal Deployment**

After metrics gathering and rearrangement, all metrics go through the Genetic algorithm. Algorithm will be implemented by using the below activities.

- population: we consider one deployment map as one chromosome. Also, the existing deployed map part of the population. Therefore, algorithms always guarantee to generate a better solution than the existing deployment map.
- crossover constraints: all the microservice must be deployed on the cluster. In each node, microservices total resource consumptions must be lower than node resource power.
- selection: more performance deployment maps are selected for the next generation. the fitness function is mainly based on performance and availability for microservice deployment.
- fitness function: If a strong dependent pair of microservices deployed in the near node or same node, it is the point that makes more score. Also, there is a scoring method for availability by counting multiple instances for a microservice.



### 3.5 System Development and Implementation

The implementation process of the proposed model will be performed as per Figure 3.2 given below.

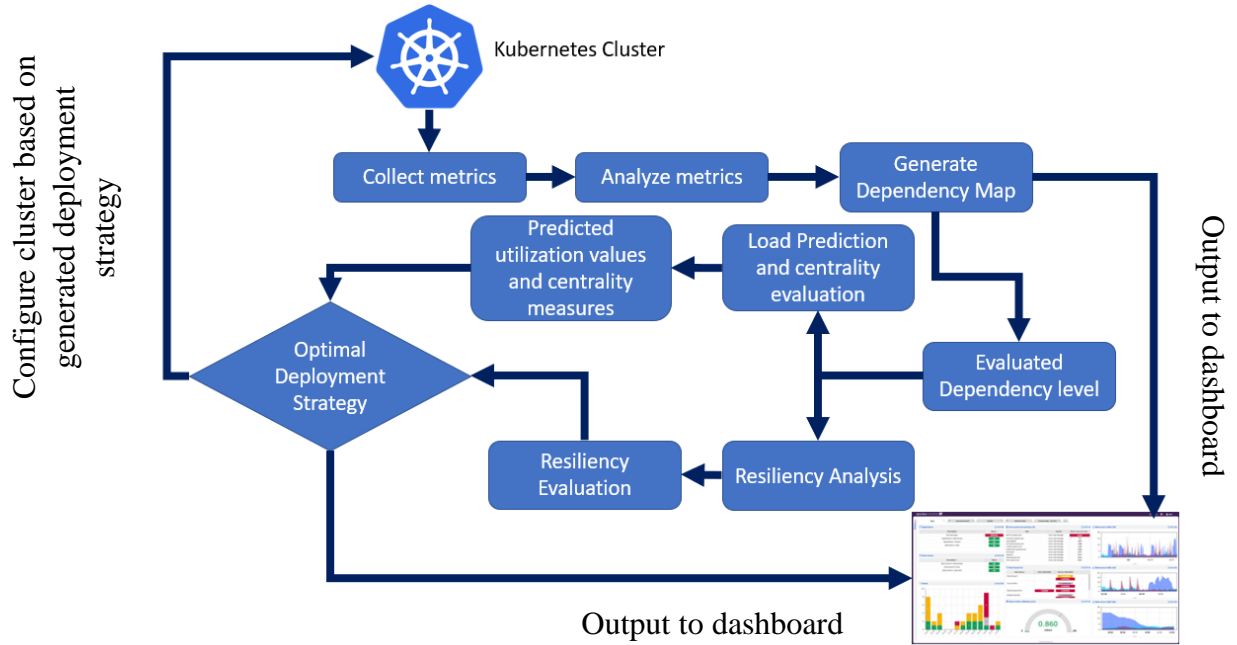


Figure 3.2: Proposed Methodology

All the obtained metrics go through the optimization algorithm. Whole system will be planned to run on master node as a single application. Admin can access to the dashboard using particular port of the master node.

### 3.5.1 Building the Final Optimization Algorithm

The genetic algorithm does not guarantee the generation of the best plan, but it always generates a better plan. After selecting the better strategy plan, the system can perform the automated deployment. To obtain node resource power and microservice resource consumption, Istio will be used. Python will be used for implementing the algorithm whereas libraries such as NetworkX be used for dealing with the node latency map. Python GA library will be used to perform the genetic algorithm part.

#### Tools

- Istio

#### Programming Languages

- Python

#### Key Python libraries expected to be used

- NetworkX
- Numpy
- GA

## **3.6 Project Requirements**

### **3.6.1 Functional Requirements**

The functional requirements for the proposed model are as follows:

- Users should be able to view the optimal deployment strategy for a given cluster.
- Users should be able to view possible deployment suggestions in order to optimize the deployment.
- The system should analyze the resiliency of the identified microservices in the cluster.
- The system should be response to the Horizontal-scaling and Vertical-scaling.
- The optimal deployment strategy for a particular deployment should be determined using the predicted load, evaluated centrality measures and resiliency evaluation.

### **3.6.2 Non-Functional Requirements**

The following are the non-functional requirements that are primarily focused during the development of the proposed model.

- Usability
- Reliability
- Availability
- Interoperability
- Performance

### 3.7 Testing

The testing phase is a key phase in the development process of this research, and it is expected to be conducted throughout the implementation process of this research project. Initial testing will be performed on a component level basis in which each component will be tested individually, and once a specific component has been developed, the testing process will begin. During component-level testing, key features and functions developed will be evaluated to ensure they are functioning as required. Once all components have been completely developed, integration testing will be performed, and the final developed model will be evaluated to ensure the overall performance is improved in the microservice cluster through the use of the proposed governance model. Here, key factors such as latency and its related measures will be used to evaluate the performance of the proposed governance model.

For this research, a test-bed of consisting of about 50 microservices is expected to be deployed in a Kubernetes cluster. Furthermore, a load-generation tool is expected to be used in order to generate sufficient metrics in order to develop the co-dependency network. These capture metrics will then be used in the load prediction and resiliency evaluation components for further analysis.

Many constraints are using to develop this optimal deployment algorithm. Before practical deployment tests, test the theoretical constraint are worked correctly or not. Especially in the fitness function, we should verify and validate fitness function scoring methods.

As an example, if the fitness function calculate score in wrong way, whole application is fails. So that go to the practical deployment testing is unnecessary thing, if fail the theoretical testing.

We hope to create a formula to evaluate the overall performance of the microservice model. This is the proposed formula:

$$\frac{\sum_{l=1}^{l=n} AVG.Res.Time_l * N_l}{\sum_{l=1}^{l=n} N_l}$$

Figure 3.7.1 performance formula

This formula calculates the average response time per request. It means that it calculates the Sum of response time of all the request divide by sum of all request in certain time period.

$l$ =fair of microservice link identifier.

$n$ =number of microservice.

$AVG.Res.Time$ =specific link average response time.

$N$ =number of requests for specific link.

We compare the performance of optimal deployment and performance of previous deployment version. This result will be guarantee that research successful or failure.

### 3.8 Time Line

The proposed timeline for the project is as follows.

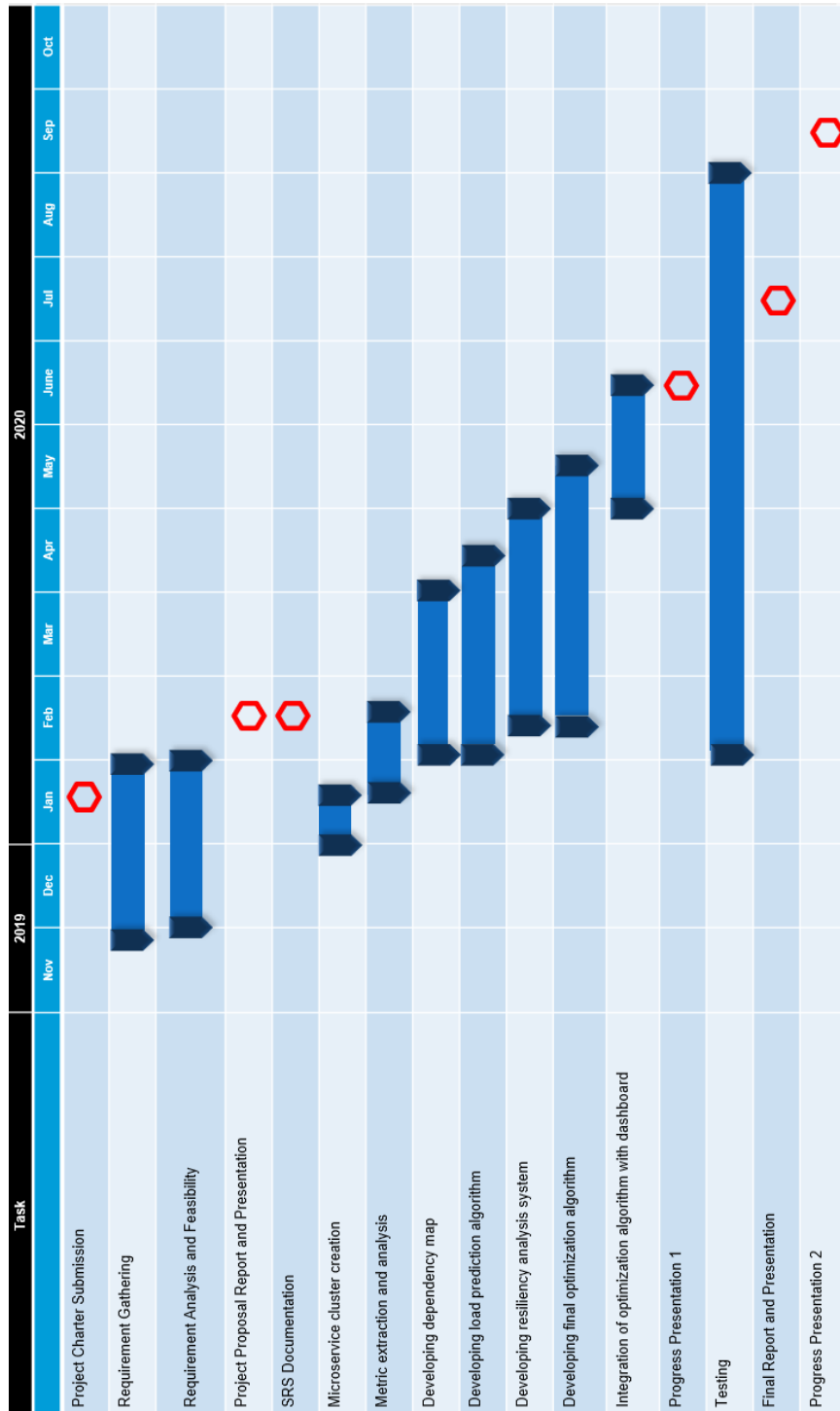


Figure 3.3: Gantt Chart

## 4.0 Personnel and Facilities

Name	Key Tasks
L.S Jayasinghe	<ul style="list-style-type: none"><li>• Retrieval of the analyzed outputs from the load prediction, dependency analysis map, as well as resiliency evaluation process.</li><li>• Formatting and data manipulation of the data received from the above-mentioned output processes, in order to be fed as inputs to the optimization algorithm.</li><li>• Development of final optimization algorithm making use of Machine Learning, which takes in inputs from the load prediction, dependency analysis map, and resiliency evaluation process, and proposes the suggested optimal deployment strategy.<ol style="list-style-type: none"><li>1. crossover constraints creation.</li><li>2. fitness function creation.</li></ol></li><li>• Integrate with the UI dashboard in order to display the optimal deployment strategy to the user.<ol style="list-style-type: none"><li>1. Optimal deployment map will be showed.</li><li>2. Suggestion list will be showed.</li></ol></li></ul>

	<ul style="list-style-type: none"> <li>Evaluate the effect of the developed optimization algorithm with respect to the performance of the microservice cluster.</li> </ul> <ol style="list-style-type: none"> <li>Theoretically evaluate the model.</li> <li>Practically evaluate the model.</li> </ol>
--	---

Table 4.1: Personnel and Resources

## 5.0 Commercialization

The commercialization of this research project is mainly considered through the development of a tool through the use of the proposed model. The developed tool will be developed as a Business Intelligence Dashboard which makes use of the proposed model to provide developers and system administrators an easy and efficient way in which to optimize their Kubernetes deployment by aiming to provide the following benefits.

- Visualize the level of inter-dependency among deployed microservices.
- Receive suggestions in potential ways to optimize the performance and configure current deployments and automatically perform deployments based on the suggestions.
- Visualize comparison of previous deployed microservice map vs currently generated map.

The developed Business Intelligence Dashboard will allow users to access all the above-mentioned features and provide a holistic view of their deployments. Hence, this tool will be mainly targeted to be marketed as an APM tool for Kubernetes deployments for system administrators and developers. Due to the wide variety of APM tools currently available in the market which are mostly free and opensource, the initial plan is to develop this dashboard into an opensource tool in order to enter the



current market space effectively. However, throughout the years, a freemium based marketing strategy will be adopted with the inclusion of additional features.

## 6.0 Budget

The main aspect of this research is primarily focused on the development of an optimization model that aids in the deployment of microservices through Kubernetes, and hence it is primarily a software-based solution with no inclusion of external hardware.

However, there will be some costs expected to be incurred, as given in the table below.

Internet use and web hosting	6000 LKR
Publication costs	2000 LKR
Stationary	2000 LKR
TOTAL	10000 LKR

Table 6.1: Budget

**Note:** Azure Student subscription with \$100 of free credit for 12 months will be used for this project. Therefore, resource creation for the VMs needed for testing and creation of the Kubernetes cluster to be used for implementation purposes could be obtained without any additional cost and hence is not included in the budget.

## **7.0 Summary**

The primary objective of this proposed research is to develop a model that aims to improve microservice governance in Kubernetes deployments through a network science-based approach.

The model is developed primarily through a microservice dependency map based on metric analysis, and performing load prediction, and resiliency evaluation on the microservices identified using the dependency map. Finally, the outputs of these components will be fed into an optimization algorithm and display an optimal deployment strategy to the user.

The proposed model will ultimately be able to provide developers and system administrators an overview of their current deployment configurations with respect to performance and aid in governing their microservice deployments such that the optimal performance is achieved.

## 8.0 References

- [1]"docker history", *Docker Documentation*, 2020. [Online]. Available: <https://docs.docker.com/engine/reference/commandline/history>. [Accessed: 24-Feb- 2020].
- [2]"How Netflix works: the (hugely simplified) complex stuff that happens every time you hit Play", *Medium*, 2020. [Online]. Available: <https://medium.com/refraction-tech-everything/how-netflix-works-the-hugely-simplified-complex-stuff-that-happens-every-time-you-hit-play-3a40c9be254b>. [Accessed: 24- Feb- 2020].
- [3]"8 Container Orchestration Tools to Know - Linux.com", *Linux.com*, 2020. [Online]. Available: <https://www.linux.com/news/8-open-source-container-orchestration-tools-know/>. [Accessed: 25- Feb- 2020].
- [4]*Vaadin.com*, 2020. [Online]. Available: <https://vaadin.com/blog/microservices-high-availability>. [Accessed: 25- Feb- 2020].
- [5]"Evaluating Critical Performance Needs for Microservices and Cloud-Native Applications", *Medium*, 2020. [Online]. Available: <https://medium.com/netifi/evaluating-critical-performance-needs-for-microservices-and-cloud-native-applications-7675c50a8460>. [Accessed: 25-Feb- 2020].
- [6]"Test Azure virtual machine network latency in an Azure virtual network", *Docs.microsoft.com*, 2020. [Online]. Available: <https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-test-latency>. [Accessed: 25- Feb- 2020].
- [7]"Azure network round trip latency statistics", *Docs.microsoft.com*, 2020. [Online]. Available: <https://docs.microsoft.com/en-us/azure/networking/azure-network-latency>. [Accessed: 25- Feb- 2020].
- [8]"Co-locate Windows Azure VMs - Azure Windows Virtual Machines", *Docs.microsoft.com*, 2020. [Online]. Available: <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/co-location>. [Accessed: 25- Feb- 2020].
- [9]"Travelling salesman problem", *En.wikipedia.org*, 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Travelling\\_salesman\\_problem](https://en.wikipedia.org/wiki/Travelling_salesman_problem). [Accessed: 25- Feb- 2020].
- [10]"Mathematical optimization", *En.wikipedia.org*, 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Mathematical\\_optimization](https://en.wikipedia.org/wiki/Mathematical_optimization). [Accessed: 25- Feb- 2020].
- [11]"Genetic algorithm", *En.wikipedia.org*, 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Genetic\\_algorithm](https://en.wikipedia.org/wiki/Genetic_algorithm). [Accessed: 25- Feb- 2020].

- [12]"Genetic Algorithms - Introduction - Tutorialspoint", *Tutorialspoint.com*, 2020. [Online]. Available: [https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_introduction.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_introduction.htm). [Accessed: 25- Feb- 2020].
- [13]M. Bravetti, S. Giallorenzo, J. Mauro, I. Talevi and G. Zavattaro, "Optimal and Automated Deployment for Microservices", *Fundamental Approaches to Software Engineering*, pp. 351-368, 2019. Available: 10.1007/978-3-030-16722-6\_21 [Accessed 25 February 2020].
- [14]"Kubernetes Scheduler", *Kubernetes.io*, 2020. [Online]. Available: <https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/>. [Accessed: 25-Feb- 2020].

## 9.0 Appendices

