



UNIVERSITÀ DEGLI STUDI DI MESSINA

DIPARTIMENTO DI INGEGNERIA

**Corso di Laurea Magistrale in
Engineering and Computer Science**

ADVANCED ALGORITHMS AND COMPUTATIONAL MODELS

PROJECT:

**Modeling a Distributed Algorithm RAFT-like
in Omnet++ Simulator**

STUDENT:

Allegra Davide Giuseppe

ANNO ACCADEMICO 2024-2025

Contents

Overview	2
1 Distributed Systems	3
1.1 Distributed Algorithms	4
1.2 Election Algorithms and Consensus Algorithms	4
1.3 RAFT	5
2 Omnet++ Project	7
2.1 Project Design	7
2.1.1 Specifications	8
2.1.2 Evolution of the system	9
2.1.3 Deployment Diagram	10
2.2 Project Implementation	12
2.2.1 System Behavior	13
2.2.2 Omnetpp.ini file	14
2.3 Simulations Results	16
2.3.1 Conclusions	23

Overview

The following Advanced Algorithms and Computational Models project was developed to simulate the behavior of a distributed algorithm in a distributed system.

The following technologies were used to develop the following project:

- Omnet++ simulator
- C++ language

The aim of the project is to model a cluster of nodes of a dynamic distributed system, in such a way that at each simulation the number of nodes is variable. A cluster in the context of computing and distributed systems is a collection of interconnected computers (also known as nodes) that work together as a single cohesive system. These nodes communicate and coordinate their actions by passing messages, allowing them to share resources and distribute workloads efficiently.

Finally, a RAFT-like consensus algorithm (similar to RAFT behavior but not RAFT) is modeled and implemented in the node cluster to elect a leader node that handles a request and obtain consensus for service deployment.

Chapter 1

Distributed Systems

A distributed system is a network of independent computers that work together as a unified system to achieve a common goal. In a distributed system, the components, referred to as nodes, are connected through a network, and they communicate and coordinate their actions by passing messages to one another. This structure provides several advantages, such as improved performance, scalability, fault tolerance, and resource sharing.

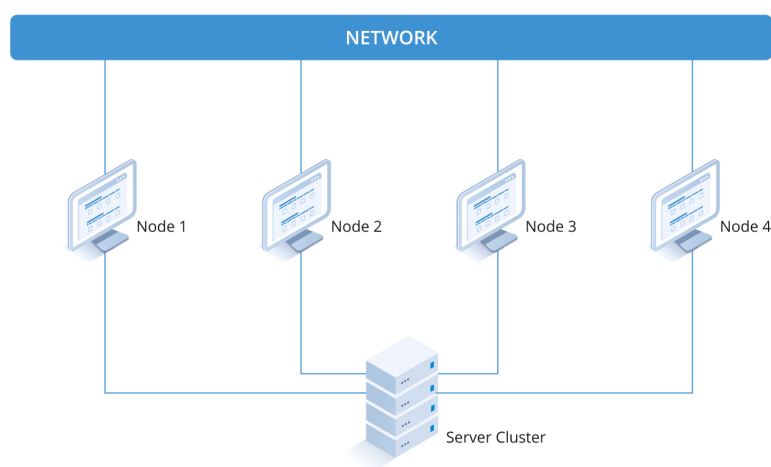


Figure 1.1

In the context of a client-server architecture within a distributed system, a client initiates a request to a server, which is part of a cluster of n interconnected nodes. Each node in the cluster can act as a server, processing the client's request and potentially coordinating with other nodes to fulfill the request efficiently.

For example, let's consider a scenario where a client sends a service re-

quest to a node in the cluster. The chosen node receives the request and may need to coordinate with other nodes to gather the required resources or data. This coordination can involve various distributed algorithms, such as leader election or consensus algorithms, to ensure that the request is handled correctly and consistently across the cluster.

The nodes within the cluster communicate through message passing, and they rely on distributed protocols to maintain the system's integrity and reliability. These protocols ensure that even if some nodes fail, the system as a whole continues to operate correctly. The redundancy and fault tolerance inherent in distributed systems make them particularly suitable for critical applications that require high availability and reliability.

1.1 Distributed Algorithms

Distributed algorithms are essential for the coordination and functioning of distributed systems. They enable multiple nodes in a network to work together seamlessly, despite potential faults or network partitions. These algorithms encompass a wide range of functionalities, including consensus protocols, leader election, and distributed data management. The effectiveness of distributed algorithms is measured by their ability to handle challenges such as latency, scalability, and fault tolerance. A well-designed distributed algorithm ensures that the system remains reliable and efficient, even in the presence of node failures or network disruptions.

1.2 Election Algorithms and Consensus Algorithms

Election algorithms are essential mechanisms in distributed systems used to coordinate activities among multiple nodes by selecting a single node to act as a coordinator or leader. This ensures that tasks are managed efficiently without conflicts. One classic example is the **Bully Algorithm**, where the node with the highest identifier becomes the leader after a series of messages exchanged among nodes. Another notable example is the **Ring Algorithm**, which arranges nodes in a logical ring and passes election messages around the ring until a leader is chosen.

1.3 RAFT

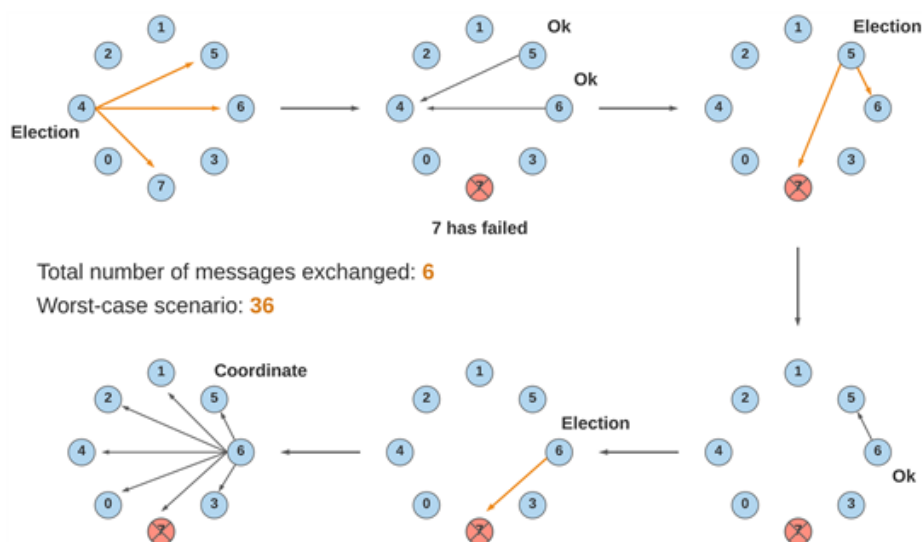


Figure 1.2

Consensus algorithms, on the other hand, are protocols used to achieve agreement on a single data value among distributed nodes, even in the presence of failures. These algorithms ensure consistency and reliability in distributed systems. Prominent examples include **Paxos** and **RAFT**.

Paxos, developed by Leslie Lamport, is a family of protocols for solving consensus in a network of unreliable processors (nodes). Despite its complexity, it guarantees that a consensus will be reached as long as a majority of nodes are operational.

RAFT is designed to be more understandable and easier to implement than Paxos. It decomposes the consensus problem into three sub-problems: leader election, log replication, and safety, making it a popular choice for implementing distributed systems. RAFT ensures that the system continues to operate correctly even if some nodes fail.

1.3 RAFT

RAFT is a consensus algorithm designed to be more understandable and easier to implement than other consensus protocols like Paxos. RAFT ensures that a group of nodes in a distributed system can agree on a single value, even in the presence of failures. The algorithm breaks down the consensus problem into three sub-problems: leader election, log replica-

1.3 RAFT

tion, and safety. RAFT elects a leader node to manage log replication and maintain the consistency of the distributed log. By simplifying the process and providing a clear state transition diagram, RAFT improves the comprehensibility and robustness of consensus mechanisms in distributed systems.

Chapter 2

Omnet++ Project



Figure 2.1

OMNeT++ is a versatile, modular, and extensible simulation environment primarily used for network simulations. It supports a wide range of applications, from protocol modeling to complex network architecture simulations. OMNeT++ is built on a component-based architecture, allowing users to create reusable modules for various network elements. Its powerful GUI tools facilitate the design, execution, and analysis of simulations. With support for C++ and a wide array of extensions and plugins, OMNeT++ is an invaluable tool for researchers and developers working on network protocols, distributed systems, and other communication-based simulations.

2.1 Project Design

Consider a fully distributed system in which each node cooperates to make orchestration decisions. These decisions are made based on a distributed consensus protocol. The consensus algorithm is inspired by RAFT.

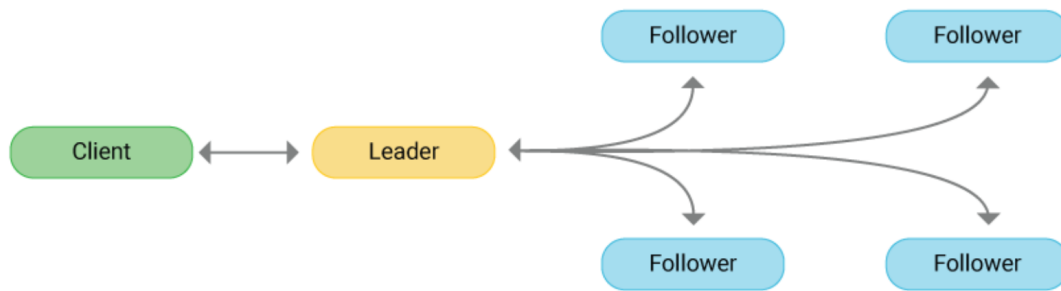


Figure 2.2

2.1.1 Specifications

A cluster system of n nodes fully connected to each other is therefore designed.

Each node is characterized by a state:

- **Node capacity:** tuple (C_ram, C_cpu) that contains information about the total resources of the node in terms of memory
- **Current workload:** tuple (W_ram, W_cpu) that contains information about the currently engaged resources of the node
- **Current number of terms:** an integer that progressively increases with each new election of a leader

Each node must host microservices:

- Each microservice is characterized by the amount of resources it requires to be executed and by the time needed for execution. The amount of resources is expressed by the tuple (R_ram, R_cpu)

Each node can be in 3 logical states for consensus:

- **Leader:** handles consensus operations in a term, and is included in the count for voting
- **Candidate:** has received an orchestration request and is applying to be the leader to handle it

- **Follower:** all other nodes during the term

The system evolves in 3 macro-phases:

- **Election and consensus**
- **Logging**
- **Deployment**

2.1.2 Evolution of the system

Evolution of the system for managing 1 orchestration request:

1. An external entity (client) sends an orchestration request to a node in the cluster (for example the closest one).
2. The node processes the request (it reads the fields: service to deploy, also the resources that are needed can be known or not, if they are not known it generates them randomly at the time of deployment).
3. The node that received the request increases the term and becomes a candidate and sends the election requests in broadcast to all the others, the election request contains the current term of the candidate and the identity of the node that is applying.
4. The responses contain: the identity, the status and the vote (boolean).
5. When a follower receives the election request, it compares the term of the candidate with its own. If the term of the candidate is higher, it updates its own and votes true. If the term of the candidate is lower it does not vote.
6. When the candidate has at least 50%+1 of the positive votes (including itself), it becomes leader and ranks the nodes based on (Figure 2.3) otherwise if not known: Nodes are classified based on workload (Figure 2.4)

7. Once the ranking is done, the leader chooses the one with the lowest usage or workload (depending on the case). And sends the vote request with the identity of the chosen node in broadcast to all the followers.
8. The followers vote again always with the term system to decide.
9. When the leader receives at least 50%+1 of the positive votes (including his own) he writes the log on the file system and performs the deployment if he is the chosen node, or forwards the message to the chosen node that performs the deployment.

$$U_i = \frac{R_{j,\text{CPU}} + W_{i,\text{CPU}}}{C_{i,\text{CPU}}} + \frac{R_{j,\text{RAM}} + W_{i,\text{RAM}}}{C_{i,\text{RAM}}}$$

Figure 2.3: Formula used if the resources of the request are known

$$\frac{W_{i,\text{CPU}}}{C_{i,\text{CPU}}} + \frac{W_{i,\text{RAM}}}{C_{i,\text{RAM}}},$$

Figure 2.4: Formula used if the resources of the request are not known

2.1.3 Deployment Diagram

The deployment diagram is a static diagram provided by the object-oriented modeling language UML to describe a system in terms of hardware resources, called nodes, and relationships between them. It is used to show how the software components are distributed with respect to the hardware resources available on the system; this diagram is built by combining the component diagram and the deployment diagram.

2.1 Project Design

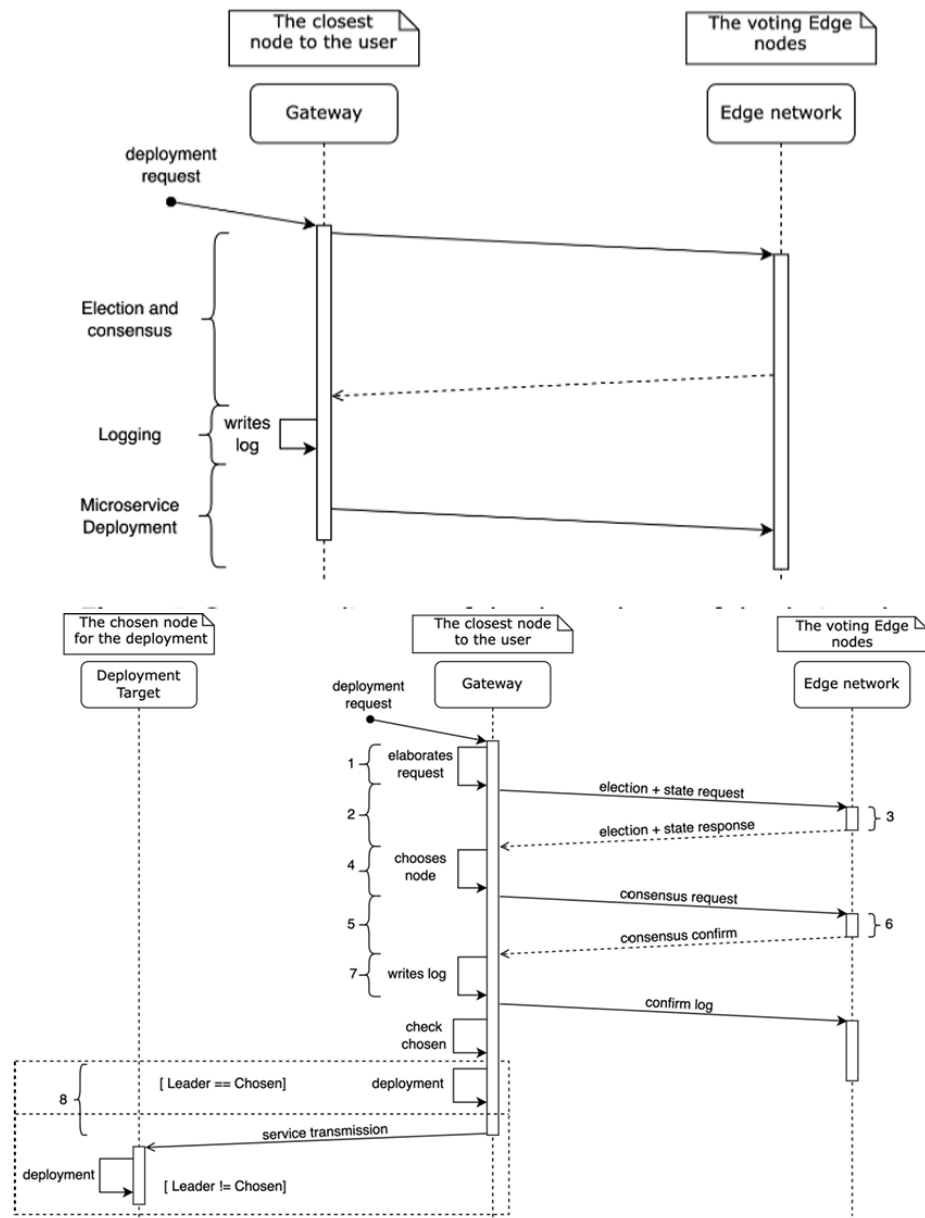


Figure 2.5: Deployment Diagram

2.2 Project Implementation

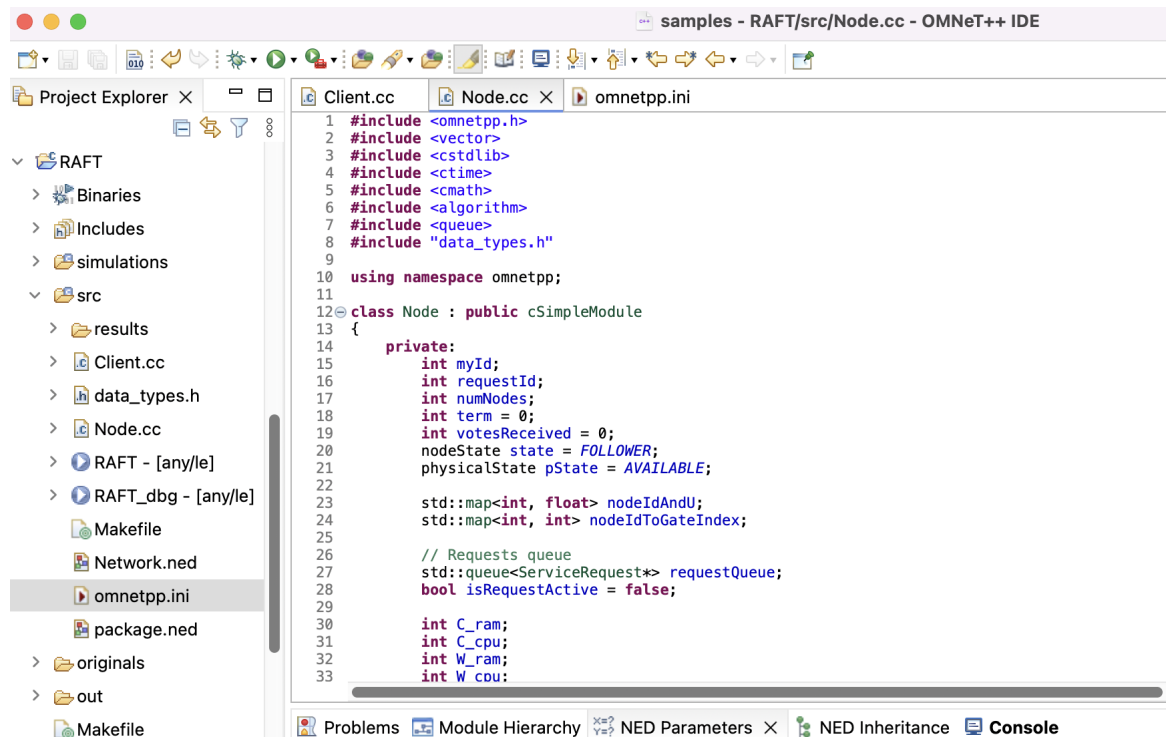


Figure 2.6: Omnet++ Project

In the main screen of Omnet it is possible to create a new project. A folder called "RAFT" has been created inside which there are some files and default folders such as the simulation and output folders. The project files are inserted inside the "src" folder.

Finally when everything is ready to be compiled the following commands are executed:

1. make clean
2. make

The project consists of the following files:

- **Network.ned:** the file containing the structure with the physical connections of the Omnet network
- **Client.cc:** It takes care of generating the number of requests per second chosen in the simulation and sends the request to a random cluster node

2.2 Project Implementation

- **Node.cc:** It is the most important logical part, it deals with managing the behavior of the nodes when a request arrives and how to manage the consensus algorithm
- **data_types.h:** It is the file that contains the main structures for transmitting messages in the network between the various nodes
- **omnetpp.ini:** the Omnet network configuration file. In this file you can describe various configurations based on the simulations you choose. You can also configure global variables

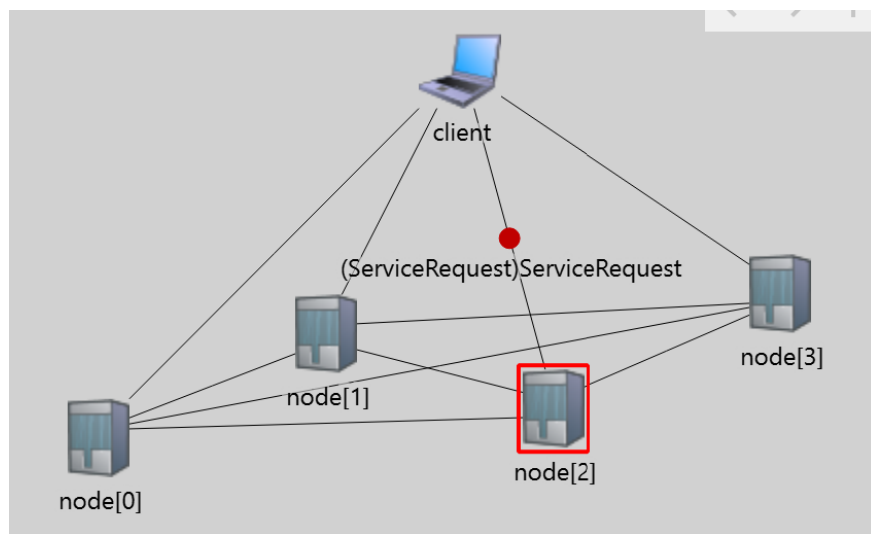


Figure 2.7: Omnet++ Project

2.2.1 System Behavior

The system behavior evolves as follows:

- The client generates a service deployment request containing a message with the resources generated at the moment
- The client sends the request to a cluster node
- The node receives the request and puts it in its local queue
- If the node's current queue contains other requests and these are still in progress, it will wait for the current request to finish
- When the node takes a request from its queue, it starts the election process and applies as leader

2.2 Project Implementation

- To be elected, the leader node must get half + 1 of the votes from the other nodes
- If it is elected leader, it will send a request to know which of all the nodes can handle the workload of the requested service
- A consensus is sent by the nodes on a node that has been chosen by the leader as the candidate to carry out the task
- If the majority has decided positively, the service will be entrusted to the chosen node
- Finally, the leader records the log on the local file system and advances its queue

2.2.2 Omnetpp.ini file

```
1 [General]
2 network = Network
3 seed-set = 0
4 sim-time-limit = 100s
5 record-eventlog = true
6
7 [Config RandomNodesAndReq]
8 description = "Simulation with random nodes and request/sec"
9 **Network.numNodes = intuniform(4, 16)
10 **Network.clientRequestsSec = intuniform(1, 10)
11
12 [Config Node4_1Req]
13 description = "Simulation with 4 nodes and 1 request/sec"
14 **Network.numNodes = 4
15 **Network.clientRequestsSec = 1
16
17 [Config Node4_5Req]
18 description = "Simulation with 4 nodes and 5 requests/sec"
19 **Network.numNodes = 4
20 **Network.clientRequestsSec = 5
21
22 [Config Node4_10Req]
23 description = "Simulation with 4 nodes and 10 requests/sec"
24 **Network.numNodes = 4
25 **Network.clientRequestsSec = 10
26
27 [Config Node8_1Req]
28 description = "Simulation with 8 nodes and 1 request/sec"
29 **Network.numNodes = 8
30 **Network.clientRequestsSec = 1
31
32 [Config Node8_5Req]
33 description = "Simulation with 8 nodes and 5 requests/sec"
34 **Network.numNodes = 8
```

2.2 Project Implementation

```
35 **Network.clientRequestsSec = 5
36
37 [Config Node8_10Req]
38 description = "Simulation with 8 nodes and 10 requests/sec"
39 **Network.numNodes = 8
40 **Network.clientRequestsSec = 10
41
42 [Config Node10_1Req]
43 description = "Simulation with 10 nodes and 1 request/sec"
44 **Network.numNodes = 10
45 **Network.clientRequestsSec = 1
46
47 [Config Node10_5Req]
48 description = "Simulation with 10 nodes and 5 requests/sec"
49 **Network.numNodes = 10
50 **Network.clientRequestsSec = 5
51
52 [Config Node10_10Req]
53 description = "Simulation with 10 nodes and 10 requests/sec"
54 **Network.numNodes = 10
55 **Network.clientRequestsSec = 10
56
57 [Config Node12_1Req]
58 description = "Simulation with 12 nodes and 1 request/sec"
59 **Network.numNodes = 12
60 **Network.clientRequestsSec = 1
61
62 [Config Node12_5Req]
63 description = "Simulation with 12 nodes and 5 requests/sec"
64 **Network.numNodes = 12
65 **Network.clientRequestsSec = 5
66
67 [Config Node12_10Req]
68 description = "Simulation with 12 nodes and 10 requests/sec"
69 **Network.numNodes = 12
70 **Network.clientRequestsSec = 10
71
72 [Config Node14_1Req]
73 description = "Simulation with 14 nodes and 1 request/sec"
74 **Network.numNodes = 14
75 **Network.clientRequestsSec = 1
76
77 [Config Node14_5Req]
78 description = "Simulation with 14 nodes and 5 requests/sec"
79 **Network.numNodes = 14
80 **Network.clientRequestsSec = 5
81
82 [Config Node14_10Req]
83 description = "Simulation with 14 nodes and 10 requests/sec"
84 **Network.numNodes = 14
85 **Network.clientRequestsSec = 10
86
87 [Config Node16_1Req]
88 description = "Simulation with 16 nodes and 1 request/sec"
89 **Network.numNodes = 8
90 **Network.clientRequestsSec = 1
91
92 [Config Node16_5Req]
```


2.3 Simulations Results

```
93 description = "Simulation with 16 nodes and 5 request/sec"
94 **Network.numNodes = 8
95 **Network.clientRequestsSec = 5
96
97 [Config Node16_10Req]
98 description = "Simulation with 16 nodes and 10 requests/sec"
99 **Network.numNodes = 8
100 **Network.clientRequestsSec = 10
```

2.3 Simulations Results

The main objective of this experimental phase was to evaluate the performance of the RAFT-inspired distributed system with respect to the handling of service deployment requests. In particular, we wanted to understand:

- Actual throughput, i.e. what percentage of incoming requests are actually processed;
- Average deployment latency, understood as the average time between the start of handling a request and its actual conclusion;
- Efficiency of the election mechanism, measured as the percentage of elections won in the first round;
- Network overhead, derived from the number of messages exchanged between nodes.

The main metrics considered were:

- Requests arrived
- Requests processed
- MessagesExchanged
- Workload of the simulation (arrived-processed)
- ElectionWonFirstRound
- TotalElections

2.3 Simulations Results

These measures provide a comprehensive view of performance against the design goal of efficiency and scalability.

General
RandomNodesAndReq -- Simulation with random nodes and request/sec
Node4_1Req -- Simulation with 4 nodes and 1 request/sec
Node4_5Req -- Simulation with 4 nodes and 5 requests/sec
✓ Node4_10Req -- Simulation with 4 nodes and 10 requests/sec
Node8_1Req -- Simulation with 8 nodes and 1 request/sec
Node8_5Req -- Simulation with 8 nodes and 5 requests/sec
Node8_10Req -- Simulation with 8 nodes and 10 requests/sec
Node10_1Req -- Simulation with 10 nodes and 1 request/sec
Node10_5Req -- Simulation with 10 nodes and 5 requests/sec
Node10_10Req -- Simulation with 10 nodes and 10 requests/sec
Node12_1Req -- Simulation with 12 nodes and 1 request/sec
Node12_5Req -- Simulation with 12 nodes and 5 requests/sec
Node12_10Req -- Simulation with 12 nodes and 10 requests/sec
Node14_1Req -- Simulation with 14 nodes and 1 request/sec
Node14_5Req -- Simulation with 14 nodes and 5 requests/sec
Node14_10Req -- Simulation with 14 nodes and 10 requests/sec
Node16_1Req -- Simulation with 16 nodes and 1 request/sec
Node16_5Req -- Simulation with 16 nodes and 5 request/sec
Node16_10Req -- Simulation with 16 nodes and 10 requests/sec

The simulations were performed on a system with n nodes and three application load scenarios:

- 1 request/sec per client
- 5 request/sec per client
- 10 request/sec per client

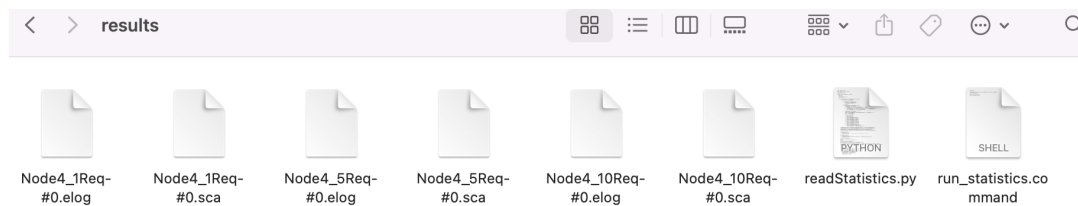
In an intermediate case, parallel deployment (from uniform(0.1, 0.5) seconds) was also tested, compared to the initial configuration with a longer delay.

Three simulations are performed one after the other. A simulation with 1 request per second, a simulation with 5 requests per second and a simulation with 10 requests per second. In this example we consider 4 nodes.

Two types of files are generated at the end of the simulation:

- Scalar .sca
- Event logs .elog

2.3 Simulations Results



Let us focus on the scalar file. This file contains scalar values collected during or at the end of the simulation, such as counters, average times, percentages. In addition, the values I decided to enter at the end of the simulation in the `finish()` function provided by Omnet++.

```
void Node::finish()
{
    recordScalar("RequestsArrived", requestsArrived);
    recordScalar("RequestsProcessed", requestsProcessed);
    recordScalar("MessagesExchanged", messagesExchanged);
    recordScalar("ElectionWonFirstRound", electionWonFirstRound);
    recordScalar("TotalElections", totalElections);

    // Calculate AverageTimeDeployService
    simtime_t sum = 0;
    int total = AverageTimeDeployService.size();
    if (total > 0) {
        while (!AverageTimeDeployService.empty()) {
            sum += AverageTimeDeployService.front();
            AverageTimeDeployService.pop();
        }
        simtime_t averageSimTime = sum / total;
        double average = averageSimTime.dbl();
        recordScalar("AverageTimeDeployService", average);
    } else {
        recordScalar("AverageTimeDeployService", -1); // No request processed
    }
}
```

The Python code for examining the data collected in the scalar files generated by Omnet++ is provided below. The programme asks the user to enter the name of the file. For example, by typing ‘Node4’ and choosing all files with initial path ‘Node4’ as the data read type, all scalars generated with this initial name are read (3 files, as the output generated files are Node4_1Req-0.sca, Node4_5Req-0.sca, Node4_10Req-0.sca).

```
1 import pprint as pp
2 import matplotlib.pyplot as plt
3 import math
4
5 aggregated_stats = {}
6
7 def process_file(fname: str) -> None:
8     par = {}
9     nodes = {}
10    scalarNodesFile = {}
```

2.3 Simulations Results

```
11
12     try:
13         with open(fname, "r", encoding="utf-8") as file:
14             for line in file.readlines():
15
16                 if fname not in scalarNodesFile:
17                     scalarNodesFile[fname] = {}
18
19                 if line.startswith("par Network.client"):
20                     lineElements = line.split()
21                     if 'numNodes' not in par and lineElements[2] == '
numNodes':
22                         par['numNodes'] = int(lineElements[3])
23                     if 'clientRequestsSec' not in par and lineElements
[2] == 'clientRequestsSec':
24                         par['clientRequestsSec'] = int(lineElements[3])
25
26                 if line.startswith("scalar Network.node["):
27                     lineElements = line.split()
28                     node = int(lineElements[1].replace("Network.node[",
""))
29                     param = lineElements[2]
30                     value = lineElements[3]
31
32                     if node not in nodes:
33                         nodes[node] = {}
34
35                     if param == 'RequestsArrived':
36                         nodes[node][param] = int(value)
37                     elif param == 'RequestsProcessed':
38                         nodes[node][param] = int(value)
39                     elif param == 'MessagesExchanged':
40                         nodes[node][param] = int(value)
41                     elif param == 'ElectionWonFirstRound':
42                         nodes[node][param] = int(value)
43                     elif param == 'TotalElections':
44                         nodes[node][param] = int(value)
45                     elif param == 'AverageTimeDeployService_Mean':
46                         nodes[node][param] = round(float(value), 5)
47                     elif param == 'AverageTimeDeployService_CI_lower':
48                         nodes[node][param] = round(float(value), 5)
49                     elif param == 'AverageTimeDeployService_CI_upper':
50                         nodes[node][param] = round(float(value), 5)
51
52                 totalRequestsArrived = sum(node.get('RequestsArrived', 0) for
node in nodes.values())
53                 totalRequestsProcessed = sum(node.get('RequestsProcessed', 0)
for node in nodes.values())
54                 messagesExchanged = sum(node.get('MessagesExchanged', 0) for
node in nodes.values())
55                 totalElections = sum(node.get('TotalElections', 0) for node in
nodes.values())
56                 totalWonFirstRound = sum(node.get('ElectionWonFirstRound', 0)
for node in nodes.values())
57
58                 electionWonFirstRoundPercentage = (totalWonFirstRound /
totalElections) * 100 if totalElections > 0 else 0
59                 handlingRequestsEfficiencyPercentage = (totalRequestsProcessed
```

2.3 Simulations Results

```

/ totalRequestsArrived) * 100 if totalRequestsArrived > 0 else 0
60
61     print()
62     print(f"=== Statistics from: {fname} ===")
63     print()
64     print(f"Nodes: {par['numNodes']}")
65     print(f"Requests/s: {par['clientRequestsSec']}")
66     print(f"Messages exchanged: {messagesExchanged}")
67     print(f"Handling requests efficiency: {
handlingRequestsEfficiencyPercentage:.2f}% ({totalRequestsProcessed
}/{totalRequestsArrived})")
68     print(f"Elections won first round: {
electionWonFirstRoundPercentage:.2f}% ({totalWonFirstRound}/{
totalElections})")
69     print()
70     pp.pprint(nodes)
71
72     config_key = fname.split('_')[1] if '_' in fname else fname
73
74     avg_times = [node.get('AverageTimeDeployService_Mean', 0) for
node in nodes.values() if node.get('AverageTimeDeployService_Mean',
0) > 0]
75     ci_uppers = [node.get('AverageTimeDeployService_CI_upper', 0)
for node in nodes.values() if node.get('
AverageTimeDeployService_CI_upper', 0) > 0]
76     ci_lowers = [node.get('AverageTimeDeployService_CI_lower', 0)
for node in nodes.values() if node.get('
AverageTimeDeployService_CI_lower', 0) > 0]
77
78     if avg_times and ci_uppers and ci_lowers:
79         avg = round(sum(avg_times) / len(avg_times), 5)
80         ci_margin = round((sum(ci_uppers) - sum(avg_times)) / len(
avg_times), 5)
81     else:
82         avg = -1.0
83         ci_margin = 0.0
84
85     aggregated_stats[config_key] = {
86         "RequestsArrived": totalRequestsArrived,
87         "RequestsProcessed": totalRequestsProcessed,
88         "MessagesExchanged": messagesExchanged,
89         "TotalElections": totalElections,
90         "WonFirstRound": totalWonFirstRound,
91         "AvgTimeDeploy": avg,
92         "AvgTimeDeploy_CI": ci_margin
93     }
94
95     except FileNotFoundError:
96         print(f">> File '{fname}' not found.")
97     except Exception as e:
98         print(f">> Read error '{fname}': {e}")
99
100 # === MAIN ===
101
102 filename = str(input("Write file name (scalar): "))
103 typeOpen = str(input("Would you open only this file or every 1/5/10 req
files? [this-all]: "))
104
```

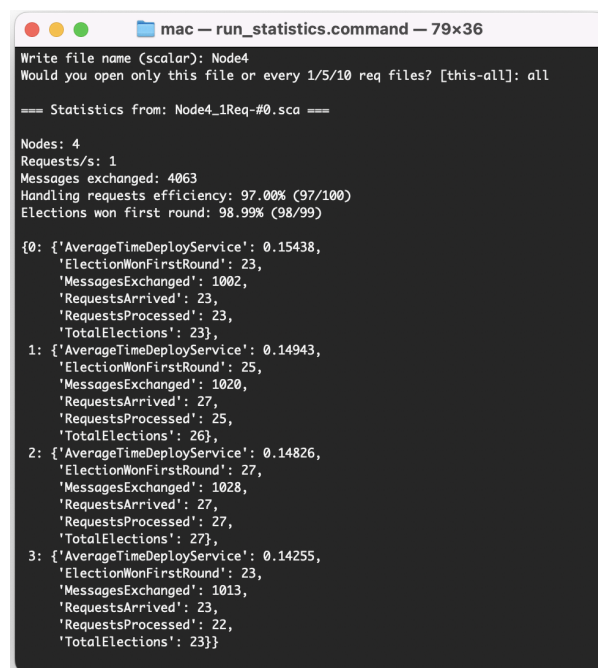
2.3 Simulations Results

```
105 if typeOpen == 'this':
106     process_file(filename)
107 else:
108     base_name = filename.split('_')[0]
109     suffixes = ["_1Req-#0.sca", "_5Req-#0.sca", "_10Req-#0.sca"]
110     for suffix in suffixes:
111         full_name = base_name + suffix
112         process_file(full_name)
113
114 # === PLOTS ===
115
116 def confidence_interval(p: float, n: int, z: float = 1.96) -> float:
117     n = int(n)
118     if n <= 0:
119         return 0.0
120     p = max(0.0, min(100.0, p))
121     p = p / 100.0
122     se = math.sqrt((p * (1 - p)) / n)
123     return z * se * 100
124
125 if aggregated_stats:
126     labels = list(aggregated_stats.keys())
127
128     messages = [aggregated_stats[k]["MessagesExchanged"] for k in
129 labels]
130     avg_time = [aggregated_stats[k]["AvgTimeDeploy"] for k in labels]
131     avg_time_ci = [aggregated_stats[k]["AvgTimeDeploy_CI"] for k in
132 labels]
133
134     handlingRequestsEfficiency_pct = []
135     handlingRequestsEfficiency_ci = []
136
137     election_pct = []
138     election_ci = []
139
140     for k in labels:
141         arrived = aggregated_stats[k]["RequestsArrived"]
142         processed = aggregated_stats[k]["RequestsProcessed"]
143         totalElections = aggregated_stats[k]["TotalElections"]
144         won = aggregated_stats[k]["WonFirstRound"]
145
146         efficiency = (processed / arrived) * 100 if arrived > 0 else 0
147         efficiency_ci = confidence_interval(efficiency, arrived)
148         handlingRequestsEfficiency_pct.append(efficiency)
149         handlingRequestsEfficiency_ci.append(efficiency_ci)
150
151         won_pct = (won / totalElections) * 100 if totalElections > 0
152         else 0
153         won_ci = confidence_interval(won_pct, totalElections)
154         election_pct.append(won_pct)
155         election_ci.append(won_ci)
156
157     fig, axs = plt.subplots(2, 2, figsize=(12, 10))
158     fig.suptitle("Omnet++ simulations statistics (RAFT-like)", fontsize
159 =16)
160
161     axs[0, 0].bar(labels, handlingRequestsEfficiency_pct, color="
162 steelblue", yerr=handlingRequestsEfficiency_ci, capsize=5)
```

2.3 Simulations Results

```
158     axs[0, 0].set_title("Handling requests efficiency (%)")
159     axs[0, 0].set_ylim(0, 110)
160     axs[0, 0].set_ylabel("% processed requests")
161
162     axs[0, 1].bar(labels, election_pct, color="mediumseagreen", yerr=
election_ci, capsize=5)
163     axs[0, 1].set_title("Elections won first round (%)")
164     axs[0, 1].set_ylim(0, 110)
165     axs[0, 1].set_ylabel("% elections won")
166
167     axs[1, 0].bar(labels, messages, color="salmon")
168     axs[1, 0].set_title("Messages exchanged (n)")
169     axs[1, 0].set_ylabel("Total messages")
170
171     axs[1, 1].bar(labels, avg_time, color="orange", yerr=avg_time_ci,
capsize=5)
172     axs[1, 1].set_title("Service deploy average time (s)")
173     axs[1, 1].set_ylabel("Seconds")
174
175     for ax in axs.flat:
176         ax.set_xlabel("Simulation configuration")
177
178     plt.tight_layout(rect=[0, 0.03, 1, 0.95])
179     plt.show()
180 else:
181     print(">> Any statistics available")
```

Clicking on the file "run_statistics.command" executes the Python file readStatistics.py. The results of the three Omnet++ simulations are printed out on the terminal. Finally, they are displayed graphically via the Python library Matplotlib.



```
mac — run_statistics.command — 79x36
Write file name (scalar): Node4
Would you open only this file or every 1/5/10 req files? [this-all]: all

=== Statistics from: Node4_1Req-#0.sca ===

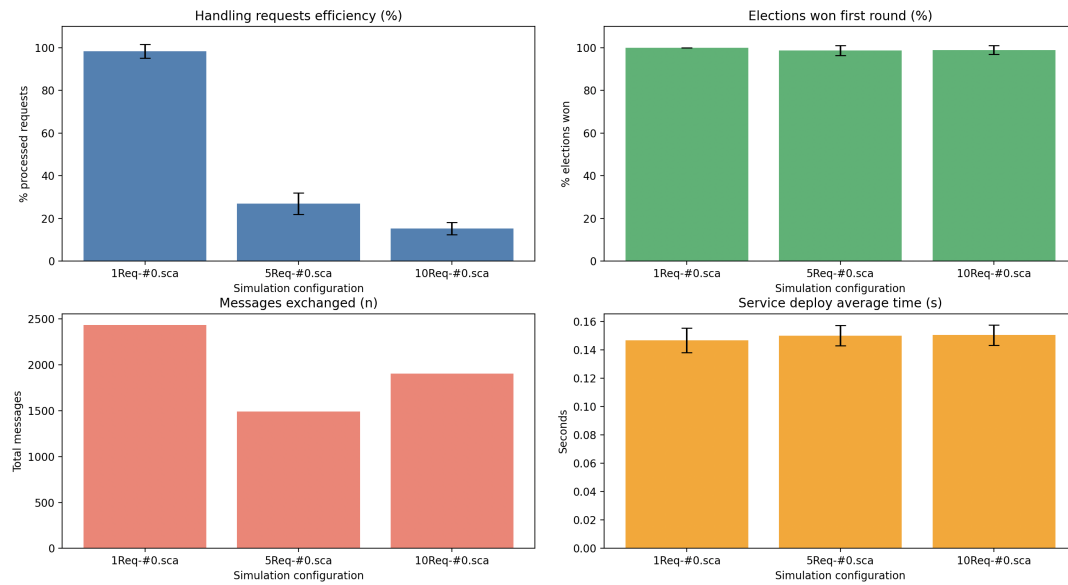
Nodes: 4
Requests/s: 1
Messages exchanged: 4063
Handling requests efficiency: 97.00% (97/100)
Elections won first round: 98.99% (98/99)

{0: {'AverageTimeDeployService': 0.15438,
'ElectionWonFirstRound': 23,
'MessagesExchanged': 1002,
'RequestsArrived': 23,
'RequestsProcessed': 23,
'TotalElections': 23},
1: {'AverageTimeDeployService': 0.14943,
'ElectionWonFirstRound': 25,
'MessagesExchanged': 1020,
'RequestsArrived': 27,
'RequestsProcessed': 25,
'TotalElections': 26},
2: {'AverageTimeDeployService': 0.14826,
'ElectionWonFirstRound': 27,
'MessagesExchanged': 1028,
'RequestsArrived': 27,
'RequestsProcessed': 27,
'TotalElections': 27},
3: {'AverageTimeDeployService': 0.14255,
'ElectionWonFirstRound': 23,
'MessagesExchanged': 1013,
'RequestsArrived': 23,
'RequestsProcessed': 22,
'TotalElections': 23}}
```

2.3 Simulations Results

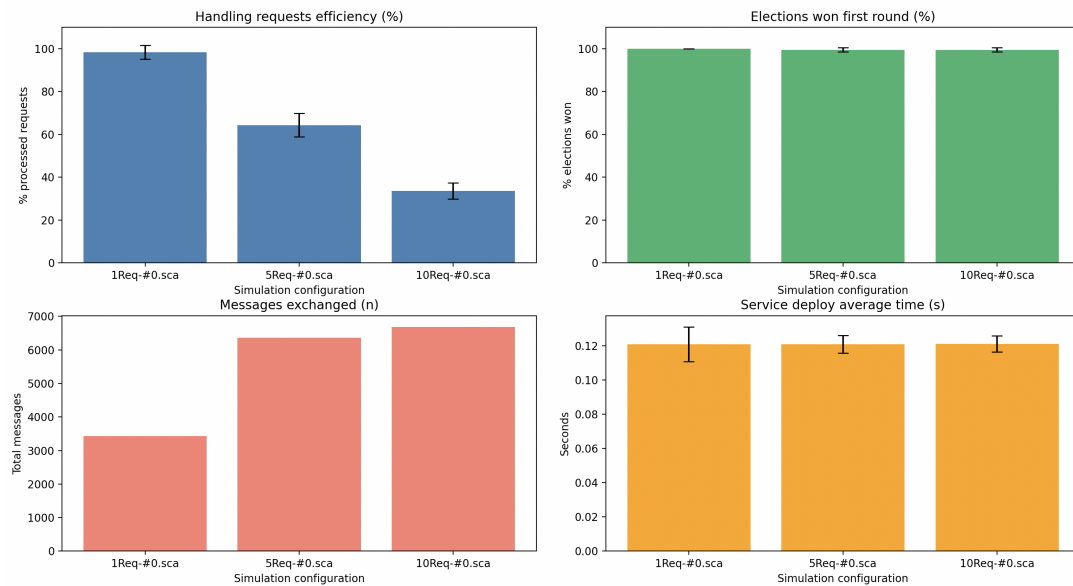
Simulation with 4 nodes

Omnet++ simulations statistics (RAFT-like)



Simulation with 8 nodes

Omnet++ simulations statistics (RAFT-like)

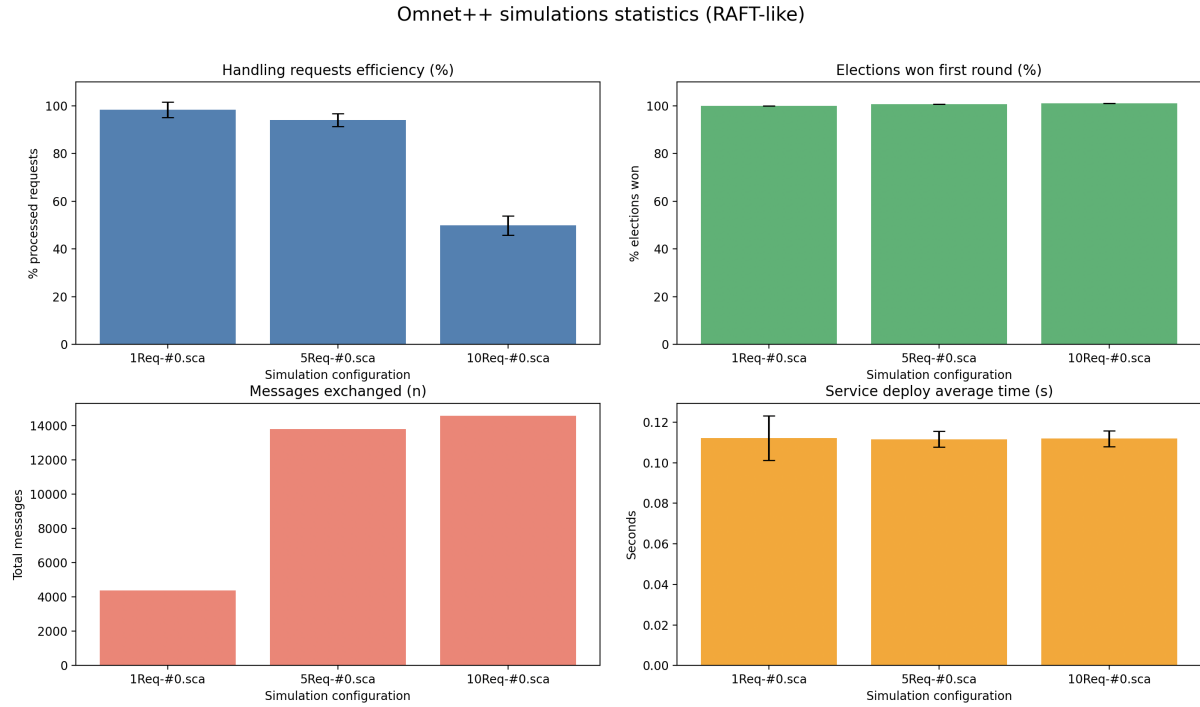


Simulation with 12 nodes

2.3.1 Conclusions

The aim of the conducted experiment was to evaluate the efficiency and scalability of a distributed coordination algorithm inspired by RAFT, in

2.3 Simulations Results



particular in the context of deploying services in a multi-node environment. Simulations were conducted on 3 scenarios with 4 nodes and different load intensities of 1, 5 and 10 requests per second.

Elections won in the first round

In all scenarios, the percentage of elections won in the first round is extremely high (over 98%), indicating that the algorithm converges quickly and without the need for re-voting. This behaviour confirms the stability of the election mechanism and the consistency of the consensus distributed even under increasing load.

Managed workload

The managed load (ratio of requests processed to requests received) shows a clear degradation with increasing intensity:

- 1 req/sec: 97%
- 5 req/sec: 15.4%
- 10 req/sec: 8.1%

This behaviour highlights a bottleneck in processing capacity: as requests increase, nodes cannot process them as efficiently. The request

2.3 Simulations Results

queue fills up and nodes do not have enough time or resources to complete the deployment of services.

Average deployment time

Despite the increasing load, the average service deployment time remains fairly stable, fluctuating between 0.14 and 0.15 seconds. This suggests that the time required to handle a single request, when processed, is not

Future improvements

The proposed system is effective in low traffic scenarios, with a high capacity to handle requests and converge quickly in the election phase. However, under high load, the processing capacity quickly becomes saturated, suggesting that:

- There may be a need to increase the number of nodes
- Optimise load management (e.g. with priority or dynamic queues)
- Introduce elastic scaling mechanisms

Looking forward, the work shows how distributed consensus can also be successfully applied in dynamic domains such as service deployment, but also highlights the importance of appropriately balancing resources and demands in real-time systems.