

**CS528**

**Reliability, IoT and Edge**

A Sahu

Dept of CSE, IIT Guwahati

## Reference:

Xie et.al, *Minimizing Redundancy to Satisfy Reliability Requirement for a Parallel Application on Heterogeneous Service-oriented Systems*, IEEE Trans. On Service Computing. 2017.

# Introduction

- Large Data Center have many generations of Server
- Bound to be heterogeneous and different failure rate
- Reliability is widely identified as
  - an increasingly relevant issue in heterogeneous service-oriented systems
  - Because processor failure affects the QoS to users
- Replication-based fault-tolerance is
  - a common approach to satisfy application's reliability requirement

# Reliability

- Reliability is defined as
  - the probability of a schedule successfully completing its execution,
  - and it has been widely identified as an increasingly relevant issue
- Fault-tolerance by primary-backup replication
  - A primary task will have 0, 1, or  $k$  backup tasks
  - is an important reliability enhancement mechanism.
- problem of minimizing redundancy
  - to satisfy reliability requirement for DAG-based parallel application
  - on heterogeneous service-oriented systems

# Reliability: Measure

- primary-backup replication scheme
  - The primary and all the backups are called replicas
- Although replication-based fault-tolerance is
  - an important reliability enhancement mechanism
  - but can not be 100% reliable in practice
- Therefore, if an APP application can
  - satisfy its specified reliability requirement
  - Named as reliability goal or reliability assurance then it is considered to be reliable
- Example: APP's reliability requirement is 0.9
  - Only if APPS 's reliability exceeds 0.9, will be reliable.

# Issue with Reliability using Redundancy

- Reliability is important QoS requirements
- Replication-based fault-tolerance
- For resource providers,
  - minimizing resource redundancy caused by replication is one of the most important concerns
- Adding more replicas
  - Could increase both reliability and
  - Redundancy for a parallel application
- Both criteria are conflicting
  - low redundancy and high reliability,
  - short schedule length and high reliability

# Application Models

- Let  $U = \{u_1, u_2, \dots, u_{|U|}\}$  represent a set of heterogeneous processors
  - where  $|U|$  is the size of set  $U$
- Parallel application running on processors is
  - Represented by a DAG  $G=(N, W, M, C)$  with known values.
- $N$  represents a set of nodes in  $G$ , and
  - Each node  $n_i \in N$  is a task with different execution time values on different processors.
- Task executions is non-preemptive

# Application Models

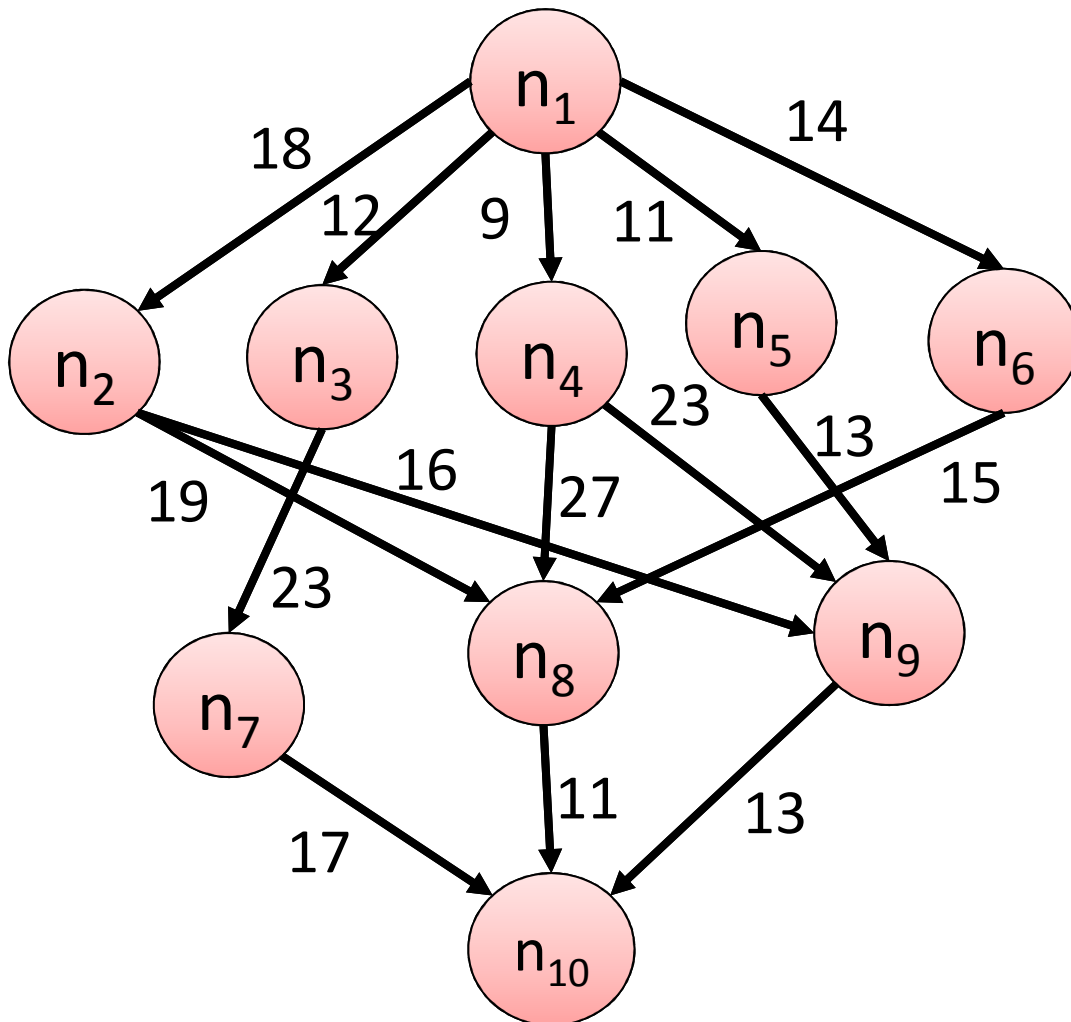
- Set of immediate predecessor tasks of  $n_i : pred(n_i)$
- set of immediate successor tasks of  $n_i : succ(n_i)$
- Tasks without predecessor tasks :  $n_{entry}$
- Tasks with no successor tasks :  $n_{exit}$
- If an application has
  - Multiple entry a dummy entry node
  - Multiple exit tasks, then a dummy exit task
  - Dummy task with zero-weight dependencies is added to the graph.
- Weight  $W$  is an  $|N| \times |U|$  matrix in which
  - $w_{i,k}$  denotes the execution time of  $n_i$  running on  $u_k$



# Application Models

- Set of communication edges is  $M$ 
  - each edge  $m_{i,j} \in M$  : communication from  $n_i$  to  $n_j$
- Communication time  $c_{i,j} \in C$  : of  $m_{i,j}$ 
  - if  $n_i$  and  $n_j$  are assigned to different processors
  - because two tasks with immediate precedence constraints need to exchange messages.
- When tasks  $n_i$  to  $n_j$  : allocated to same processor,
  - $c_{i,j}$  becomes zero because
  - Intra-processor communication cost is negligible

# Application Models: Example



Task	$u_1$	$u_2$	$u_3$
$n_1$	14	16	9
$n_2$	13	19	18
$n_3$	11	13	19
$n_4$	13	8	17
$n_5$	12	13	10
$n_6$	13	16	9
$n_7$	7	15	11
$n_8$	5	11	14
$n_9$	18	12	20
$n_{10}$	21	7	16

# Reliability Model

- Two major types of failures
  - transient failure (also called random hardware failure)
  - permanent failure
- Once a permanent failure occurs
  - Processor cannot be restored unless by replacement.
- Transient failure appears for a short time
  - and disappear without damage to processors
- Mainly takes the transient failures into account in our consideration

# Reliability Model

- Suppose you are going from IITG to Airport
  - **CAR M** : 1990 model Maruti 800 car, car is around 30 year old and chances of failure is high
  - **CAR B** : BMW 5 series 2020 car, new cars, chances of failure is low
- Occurrence of failure is dependent on
  - Failure rate of CAR and Distance travelled
- I will prefer not to go above 1km in Car M
- Reliability: Exponential of Failure Rate and Distance  $R(F, D) = e^{-F.D}$

# Reliability Model

- Occurrence of transient failure for a task follows
  - Poisson distribution
- Reliability of an event in unit time  $t$  is

$$R(t) = e^{-\lambda \cdot t}$$

- where  $\lambda$  constant failure rate per time unit of processor
- Let  $\lambda_k$  is failure rate of the processor  $u_k$
- Reliability of  $n_i$  executed on  $u_k$  in its execution time

$$R(n_i, u_k) = e^{-\lambda_k w_{ik}}$$

- Failure probability for  $n_i$  without using the active replication is  $1 - R(n_i, u_k) = 1 - e^{-\lambda_k w_{ik}}$

# Reliability with Replicas

- Each task has replicas with the active replication
- Let  $num_i$  ( $num_i \leq |U|$ ) : number of replicas of  $n_i$ .
  - Replica set of  $n_i$  is  $\{n_i^1, n_i^2, \dots, n_i^{num_i}\}$
  - where  $n_i^1$  is the primary and others are backups.
- As long as one replica of  $n_i$  successfully completed
  - then recognize, there is no occurrence of failure for  $n_i$
- Reliability of  $n_i$  is updated to

$$R(n_i) = 1 - \prod_{x=1}^{num_i} 1 - R(n_i^x, u_{pr}(n_i^x))$$

where  $u_{pr}(n_i^x)$  represents assigned processor of  $n_i^x$

- Reliability of the parallel application with precedence-constrained tasks  $R(G) = \prod_{n_i \in N} R(n_i)$

# Problem Statement

- Assign replicas and corresponding processors for each task,
  - While minimizing the number of replicas and
  - Ensuring that obtained reliability of application  $R(G)$
  - Satisfies application's reliability requirement  $R_{req}(G)$
- Find the replicas and processor assignments of all tasks to minimize

$$NR(G) = \sum_{n_i \in N} num_i$$

$$\text{— subject to } R(G) = \prod_{n_i \in N} R(n_i) > R_{req}(G)$$

$$\forall i : 1 \leq i \leq |N|$$

# Solution Approaches: Task Prioritizing

- Fault-tolerant scheduling generally consists of
  - 1) task prioritizing,
  - 2) processor selection, and
  - 3) task execution.
- Therefore, first compute task priority before processor selection
- Upward rank  $rank_u$  of a task as task priority standard
- Tasks are ordered : descending order of  $rank_u$

$$rank_u(n_i) = \overline{w}_i + \max_{n_j \in succ(n_i)} \{ c_{ij} + rank_u(n_j) \}$$

- Where  $\overline{w}_i$  is average execution time of  $n_i$



# Solution Approaches: MaxRE

- As application reliability is
  - Product of all the task reliability values
- So such problem is usually solved
  - by transferring application's reliability requirement to the sub-reliability requirements of tasks
- Sub-reliability requirement for each task

$$R_{req}(n_i) = \sqrt[N]{R_{req}(G)}$$

- If sub-reliability requirement of each task can be satisfied by active replication  $R(n_i) \geq R_{req}(n_i)$
- Main idea is to iteratively select
  - Replica  $n_i^x$  and processor  $u_{pr}(n_i^x)$  with the maximum  $R(n_i^x, u_{pr}(n_i^x))$  until  $R(n_i) \geq R_{req}(n_i)$

# Solution Approaches: MaxRE

- As application reliability is
  - Product of all the task reliability values
- So such problem is usually solved
  - by transferring application's reliability requirement to the sub-reliability requirements of tasks
- Sub-reliability requirement for each task

$$R_{req}(n_i) = \sqrt[N]{R_{req}(G)}$$

- If sub-reliability requirement of each task can be satisfied by active replication  $R(n_i) \geq R_{req}(n_i)$
- Main idea is to iteratively select
  - Replica  $n_i^x$  and processor  $u_{pr}(n_i^x)$  with the maximum  $R(n_i^x, u_{pr}(n_i^x))$  until  $R(n_i) \geq R_{req}(n_i)$

# MaxRE: Example

- Suppose  $R_{req}(G) = 0.9$  and 4 tasks in DAG
- Sub-reliability requirement for each task

$$R_{req}(n_i) = \sqrt[4]{0.9} = 0.97401$$

- Each task need to satisfy the reliability above 0.97401

# Solution Approaches: RR

- Demerits of MaxRE
  - Sub-reliability requirements of all tasks are equal and high,
  - such that it needs more replicas with extra redundancy
  - to satisfy the sub-reliability requirement of each task
- Unequal task sub-reliability may yield good solution
  - **Start with equal reliability**
- Lower down the sub-reliability requirement
  - of tasks while still satisfying the application's reliability requirement

# Solution Approaches: RR

- First, the sub-reliability requirement
  - for entry task is  $R_{req}(n_i) = \sqrt[|N|]{R_{req}(G)}$
- But for the next sequence of task can
  - calculated based on required and already archived

$$R_{req}(n_{seq(j)}) = \sqrt[|N|-j+1]{\frac{R_{req}(G)}{\prod_{x=1}^j R(n_{seq(x)})}}$$

## RR: Example

- Suppose  $R_{req}(G) = 0.9$  and 4 tasks in DAG
- Sub-reliability requirement for First task

$$R_{req}(n_i) = \sqrt[4]{0.9} = 0.97401$$

- Suppose we allocate 5 replica to satisfy and we achieved a bit high reliability  $0.985 > 0.97401$
- We can use this higher value to reduce the reliability for other remaining tasks

$$R_{req}(n_i) = \sqrt[3]{0.9/0.985} = 0.97 < 0.97401$$

# Enough Replication for redundancy minimization

- Demerit of RR
  - Reduction ranges of tasks near entry task are much lower than those near to the exit task
  - actual sub-reliability requirements show unfairness among tasks
  - still requires unnecessary redundancy to satisfy application's reliability requirement.
- If one task has  $R(n_i) < R_{\text{req}}(G)$ , then
  - no matter how many replicas for any other tasks
  - $R_{\text{req}}(G)$  cannot be satisfied

# Lower Bound on Redundancy

- Lower bound (LB) on  $R_{req}(n_i)$  for any task  
 $R_{lbreq}(n_i) = R_{req}(G)$  and  $R_{req}(n_i) \geq R_{lbreq}(n_i)$
- LB on number of replicas for task  $n_i$  to satisfy

$$1 - \prod_{x=1}^{lb(n_i)} \left( 1 - R(n_i^x, u_{pr}(n_i^x)) \right) \geq R_{lbreq}(n_i)$$



# Calculation of LB replication

- Steps to select the replica and corresponding processor with minimum number of replicas
- Steps 1: Calculate  $R(n_i, u_k)$  of each task
  - on all available processors
  - if a replica of  $n_i$  has been assigned to  $u_k$  processor then  $u_k$  is unavailable for  $n_i$
- Step 2: to minimize the number of replicas
  - Select the replica  $n_i^x$  of task  $n_i$  and
  - corresponding processor  $u_{pr(n_i^x)}$  with the maximum  $R(n_i^x, u_{pr(n_i^x)})$
- Step 3: repeat Step 1 & 2 to satisfy  $R_{lbreq}(n_i)$

# Adding Enough Replication

- Using LBR all the tasks merely satisfy

$$R_{lbr}(n_i) = R_{req}(G)$$

- We should add more new replicas for tasks to satisfy application's reliability requirement.
- Choosing remaining replicas is complex work
  - because different replicas of different tasks
  - may cause different reliability values on different processors

# Adding Enough Replica

- Given
  - the current number of replicas for  $n_i$  is  $h = \text{num}_i$
  - and the application reliability is  $R(G)$
- if a new replica  $n_i^{h+1}$  is assigned
  - to the processor  $u_k = u_{pr}(n_i^{h+1})$  for  $n_i$ , then the number of replicas is changed to  $h+1$
- The task reliability is changed to

$$R_{new}(n_i) = 1 - \prod_{x=1}^{h+1} (1 - R(n_i^x, u_{pr}(n_i^x)))$$

- The application reliability due  $R_{new}(n_i)$  is changed to

$$R^i(G) = R_{new}(n_i) * \prod_{n_j \in N, i \neq j} R(n_j)$$

# Adding Enough Replica

- **Step 1: Each available task**
  - Available: if replicas of a task have not been assigned to all the processors
  - Assumed to be replicated once on an available processor with maximum  $R(n_i, u_k)$  and calculate  $R_{\text{new}}(n_i)$
- **Step 2: Calculate APP reliability  $R_i(G)$  of each task**
- **Step 3: Select replica  $n^x_i$  and processor *with***
  - maximum  $R_i(G)$  from all the generated replicas
$$R_i(G) = \max \{ R_1(G), R_2(G), \dots, R_{|N|}(G) \}$$
- **Step 4: Repeat Steps (1), (2), and (3)**
  - until  $R(G) > R_{\text{req}}(G)$  is satisfied

# Heuristics Replication for Redundancy Minimization (HRRM)

- Similar to RR Approach
  - unassigned tasks can also be pre-supposed as assigned tasks with known reliability values
  - Pre-supposed value: can be upper bound value

$$R_{ubreq}(n_i) = \sqrt[|N|]{R_{req}(G)}$$

- Heuristic: Assume the task to be assigned is  $n_{seq(j)}$ 
  - Ensure :  $R_{req}(G)$  is satisfied at each task assignment

$$R_{req}(n_{seq(j)}) = \frac{R_{req}(G)}{\prod_{x=1}^{j-1} R(n_{seq(x)}) \times \prod_{y=j+1}^{|N|} R_{ubreq}(n_{seq(y)})}$$

# Edge Computing and IoT

# Edge Computing

- Edge computing
  - Brings computation and data storage closer to the sources of data
  - This is expected to improve response times and save bandwidth
- Edge computing is a topology- and location-sensitive form of distributed computing
- IoT uses the Edge Computing

# Edge Computing

- The origins lie
  - in **content distributed networks (CDN)**
  - that were created in the late 1990s
  - to serve web and video content from edge servers that were deployed close to user
- Example: Netflix, Utube
- CDN focus on Data not Processing
- Edge Computing focus on both Processing and Data



# IoT

- Describes physical objects (or groups objects)
  - with sensors, processing ability, software, and other technologies
  - that connect and exchange data with other devices and systems over the Internet or other communications networks
  - Do a whole one collaborative work
- Crowd sourcing Google map traffic, Building/Campus for energy efficiency

# IoT

- IoT can be used for Smart home, Smart Speaker, Elder Care
- Smart health care, Manufacturing, Transportation
- Agriculture, Environmental Monitoring
- Internet of Battle field of things, Ocean of Things

# IoT : Characteristics

- Source of Data : Sensors, Camera, Microphone, Monitors
- Sources may be mobile, ubiquitous, battery powered, all may not be directly connected to cloud
- Collection of Data and Storing of data
  - Network, Delay
  - How to send data to server for storing efficiently
- Processing of Data
  - Resource management
- Action based on Decision on Data Processing
- Many IoTs action : timely manner

# **User App Data Distribution in Edge Computing**

**Cost-Effective App Data Distribution in Edge Computing, IEEE Trans Parallel Dist. System, Jan 21**

# Edge Data Distribution: Intro

- Cloud computing is the practice of
  - using a network of remote servers hosted on the internet
  - to store, manage, and process data,
  - rather than a local server or a personal computer.
- Latency influenced by
  - the number of router hops, packet delays
  - introduced by virtualization in the network,
  - or the server placement within a data center,
  - has always been a key issue for cloud migration.
- Other conventional network paradigms
  - cannot handle the huge increase in the network latency and congestion
  - caused by the resources at the edge of the cloud.

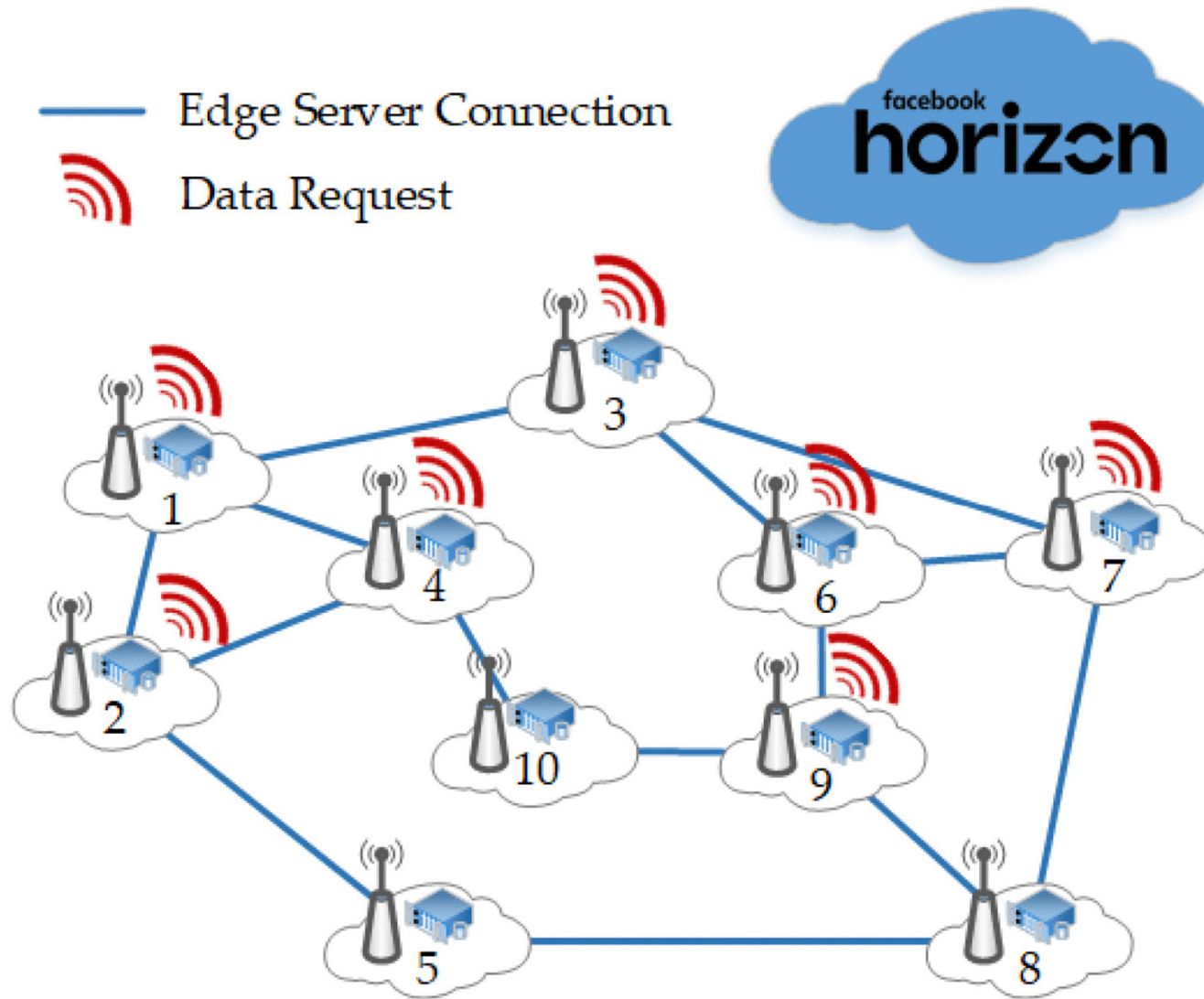
# Edge Data Distribution: Intro

- edge computing
  - which is essentially the process of decentralizing the computer services
  - with the help of edge servers deployed at the base stations.
- From an app vendor's perspective,
  - caching data on edge servers
  - considerably reduce both the latency for their users to fetch data and the volume of their app data transmitted between the cloud and its users.
  - Thus, reducing the transmission costs.

# An Industry Example: Facebook Horizon

- Facebook Horizon: VR application
- Facebook Horizon can benefit greatly
  - from distributing most popular VR videos and VR games onto the edge servers.
- VR applications are very latency sensitive
  - thus caching these data onto edge servers
  - will increase VR performance, experience and sensitivity.
- Cost-Ineffective app data distribution
  - Can thereby cost Facebook Horizon significantly more.

# An Industry Example: Facebook Horizon

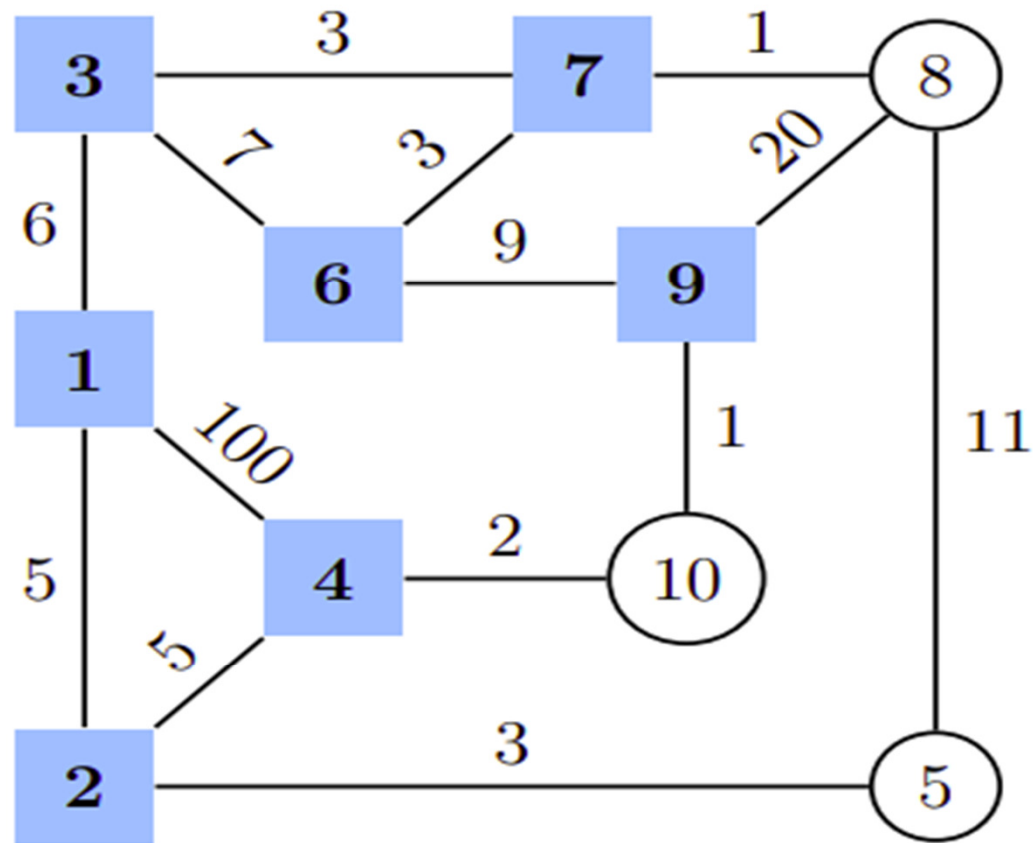




# Problem Definition: EDD

- $N$  edge servers in a particular area and model as graph  $G$ .
  - For each edge server  $v$ , graph  $G$  has a node  $v$ .
  - For each of the linked edge servers  $(u,v)$ , graph  $G$  has a corresponding edge  $e_{(u,v)}$ .
- $G(V,E,W)$  to represent the graph
  - where  $V$  is the set of nodes or edge servers in the graph,
  - $E$  are the set of edges in the graph, and
  - $W$  is the set of weights corresponding to the edges.
- Let  $R$  denote the set of destination edge servers in graph  $G$ .

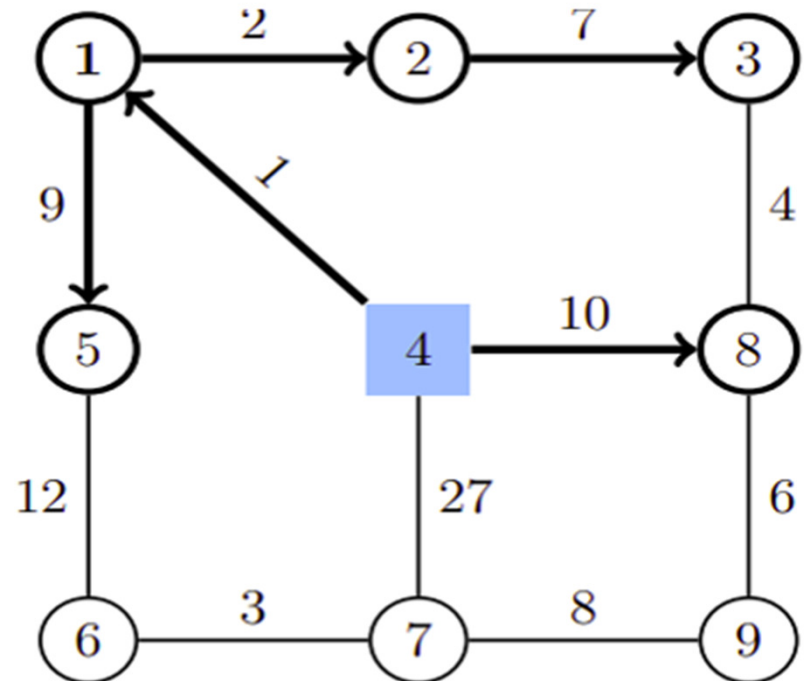
# Problem Definition: EDD



**Fig. 4.1:** EDD scenario with 10 edge servers

# Problem Definition : EDD

- Let  $L_{limit}$  be the vendors' EDD length constraint.
- Two Possible Scenario – C2E and E2E.
- We define a ratio  $\lambda$  to specify the constant weight for the cloud to edge server edges.



EDD example to demonstrate  $L_{limit}$

# Problem Definition: EDD

$$s_v = \begin{cases} 1, & \text{if } v \text{ is an initial transit edge server} \\ 0, & \text{if } v \text{ is not an initial transit edge server} \end{cases} \quad (1)$$

$$\tau_{(u, v)} = \begin{cases} 1, & \text{if data is transmitted through } e_{(u, v)} \\ 0, & \text{if data is not transmitted through } e_{(u, v)} \end{cases} \quad (2)$$

$$Connected(S, T, u, v) = true, \forall v \in R, \exists u, s_u = 1 \quad (3)$$

$$P_{\text{delay}} = \frac{L_{\text{link}}}{s_{\text{medium}}} \quad (4)$$

$$W_{(c, v)} = \lambda, \forall v \in V \setminus \{c\} \quad (5)$$

$$0 \leq L_v \leq L_{\text{limit}}, L_v \in Z^+, \forall v \in R \quad (6)$$

$$\text{minimize } (Cost_{\text{C2E}}(S) + Cost_{\text{E2E}}(T)) \quad (7)$$