

# 15-440

# Distributed Systems

# Recitation 9

Tamim Jabban

# Project 3

- Involves using the **Message Passing Interface (MPI)**
- The Project will apply MPI to the **popular clustering** problem
- The clustering problem will be solved via the **K-Means algorithm**
- **Due date:** November 12<sup>th</sup>

# What is MPI?

- MPI = Message Passing Interface
- MPI is a library of routines that can be used to create parallel programs.

# Fundamentals: *Communicators & Groups*

- MPI defines **communicators and groups** to define which collection of processes may communicate with each other
- Most MPI routines/functions require a *communicator* as an input parameter
- For simplicity, we'll be using the **MPI\_COMM\_WORLD** communicator
  - This communicator includes *all of your MPI processes*

# Fundamentals: *Ranks*

- Within a communicator, each process has its own and **unique ID** or *rank*
  - These IDs are commonly used conditionally to control program execution
- Ranks start from **0**

# MPI Routines

- `MPI_Init(int *argc, char ***argv)`
- This initializes the MPI execution environment.
  - Therefore, this **must** be called (once) **at the start of every MPI program**

# MPI Routines

- `MPI_Comm_size(MPI_Comm comm, int *size)`
- This determines the number of processes in the group associated with the `comm` communicator

# MPI Routines

- `MPI_Comm_rank(MPI_Comm comm, int *rank)`
- This determines the **rank** of the calling process within the communicator.



# MPI Routines

- `MPI_Wtime()`
- This returns an elapsed wall clock time in seconds (double precision) on the calling processor.
  - We'll use this to **measure the runtime** of an MPI program

# MPI Routines

- `MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm )`
- This is a basic **blocking send** operation. It returns only after the application has sent the data to the recipient(s)

# MPI Routines

- `MPI_Recv( void *buf, int count, MPI_Datatype datatype, int src, int tag, MPI_Comm comm, MPI_Status *status )`
- This receives a message and blocks until the requested data is available in the application buffer

# MPI Routines

- **MPI\_Finalize()**
- This terminates the MPI execution environment.
  - This should be called **at the end** of every MPI program

# Using MPI

- 4 VMs/nodes provisioned
- Coding in C
- Using n01 as your primary VM

# Running MPI

- Machinefile
- Compiling:
  - `mpicc HelloWorld.c -o HelloWorld`
- Copying object file:
  - `scp -p "HelloWorld" andrewid-n02.qatar.cmu.local:/home/hadoop/`
- Running the program:
  - `mpiexec -f machinefile -n 2 ./HelloWorld`

# MPI Examples

- Together, we'll program two MPI examples:
  - HelloWorld
  - A Distributed Sum Program