Scheduling in Distributed System

Distributed Scheduling

Introduction

- Good resource allocation schemes are needed to fully utilize the computing capacity of the DS
- Distributed scheduler is a resource management component of a DOS
- It focuses on judiciously and transparently redistributing the load of the system among the computers
- Target is to maximize the overall performance of the system
- More suitable for DS based on LANs

Motivation

- A locally distributed system consists of a collection of autonomous computers connected by a local area communication network
- Users submit tasks at their host computers for processing
- Load distributed is required in such environment because of random arrival of tasks and their random CPU service time
- There is a possibility that several computers are heavily loaded and others are idle of lightly loaded
- If the load is heavier on some systems or if some processors execute tasks at a slower rate than others, this situation will occur often

Distributed Systems Modeling

- Consider a system of N identical and independent servers
- Identical means that all servers have the same task arrival and service rates
- Let ρ be the utilization of each server, than P=1- ρ, is the probability that a server is idle
- If the ρ=0.6, it means that P=0.4,
- If the systems have different load than load can be transferred from highly loaded systems to lightly load systems to increase the performance

Issues in Load Distribution

- Load
  - Resource queue lengths and particularly the CPU queue length are good indicators of load
  - Measuring the CPU queue length is fairly simple and carries little overhead
  - CPU queue length does not always tell the correct situation as the jobs may differ in types
  - Another load measuring criterion is the processor utilization
  - Requires a background process that monitors CPU utilization continuously and imposes more overhead
  - Used in most of the load balancing algorithms

Classification of LDA

- Basic function is to transfer load from heavily loaded systems to idle or lightly loaded systems
- These algorithms can be classified as :
  - Static
    - decisions are hard-wired in the algorithm using a prior knowledge of the system
  - Dynamic
    - use system state information to make load distributing decisions
  - Adaptive
    - special case of dynamic algorithms in that they adapt their activities by dynamically changing the parameters of the algorithm to suit the changing system state

Basic Terminologies

- Load Balancing vs. Load sharing
  - Load sharing algorithms strive to reduce the possibility for a system to go to a state in which it lies idle while at the same time tasks contend service at another, by transferring tasks to lightly loaded nodes
  - Load balancing algorithms try to equalize loads at al computers
  - Because a load balancing algorithm transfers tasks at higher rate than a load sharing algorithm, the higher overhead incurred by the load balancing algorithm may outweigh this potential performance improvement

Basic Terminologies (contd.)

- Preemptive vs. Non-preemptive transfer
  - Preemptive task transfers involve the transfer of a task that is partially executed
  - Non-preemptive task transfers involve the transfer of the tasks that have not begun execution and hence do not require the transfer of the task’s state
  - Preemptive transfer is an expensive operation as the collection of a task’s state can be difficult
  - What does a task’s state consist of?
  - Non-preemptive task transfers are also referred to as task placements

Components of a Load Balancing Algorithm

- Transfer Policy
  - determines whether a node is in a suitable state to participate in a task transfer
  - requires information on the local nodes’ state to make decisions
- Selection Policy
  - determines which task should be transferred
- Location Policy
  - determines to which node a task selected for transfer should be sent
  - requires information on the states of remote nodes to make decisions
- Information policy
  - responsible for triggering the collection of system state information
  - Three types are: Demand-Driven, Periodic, State-Change-Driven

Stability

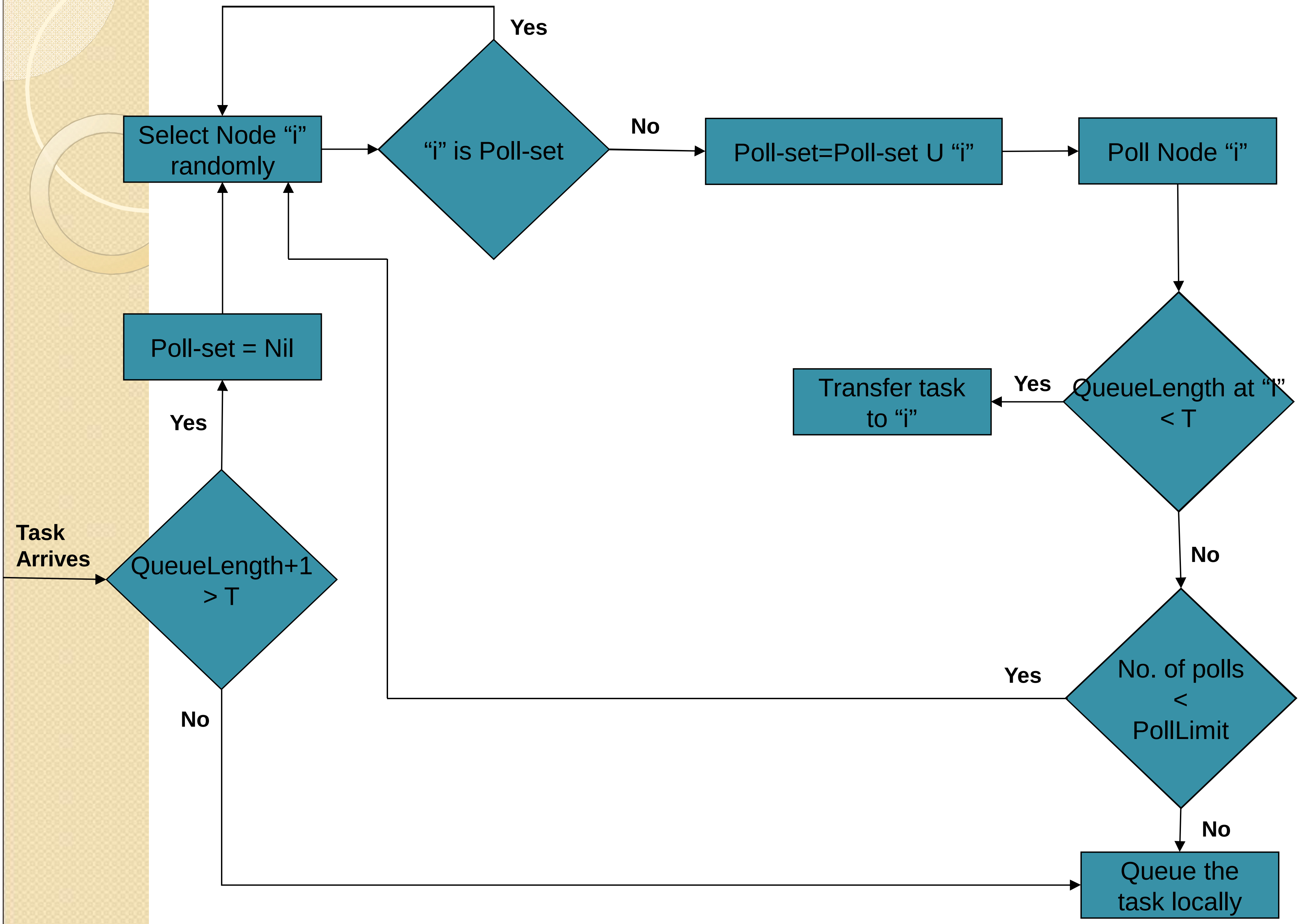
- The two views of stability are,
  - The Queuing-Theoretic Perspective
    - A system is termed as unstable if the CPU queues grow without bound when the long term arrival rate of work to a system is greater than the rate at which the system can perform work.
  - The Algorithmic Perspective
    - If an algorithm can perform fruitless actions indefinitely with finite probability, the algorithm is said to be unstable.

Load Distributing Algorithms

- Sender-Initiated Algorithms
- Receiver-Initiated Algorithms
- Symmetrically Initiated Algorithms
- Adaptive Algorithms

Sender-Initiated Algorithms

- Activity is initiated by an overloaded node (sender)
- A task is sent to an underloaded node (receiver)
  - Transfer Policy
    - A node is identified as a sender if a new task originating at the node makes the queue length exceed a threshold T.
  - Selection Policy
    - Only new arrived tasks are considered for transfer
  - Location Policy
    - Random: dynamic location policy, no prior information exchange
    - Threshold: polling a node (selected at random) to find a receiver
    - Shortest: a group of nodes are polled to determine their queue
  - Information Policy
    - A demand-driven type
  - Stability
    - Location policies adopted cause system instability at high loads



Task Arrives

QueueLength+1 > T

Yes

Poll-set = Nil

Select Node “i” randomly

“i” is Poll-set

Yes

No

Poll-set=Poll-set U “i”

Poll Node “i”

QueueLength at “i” < T

Yes

Transfer task to “i”

No

No. of polls < PollLimit

Yes

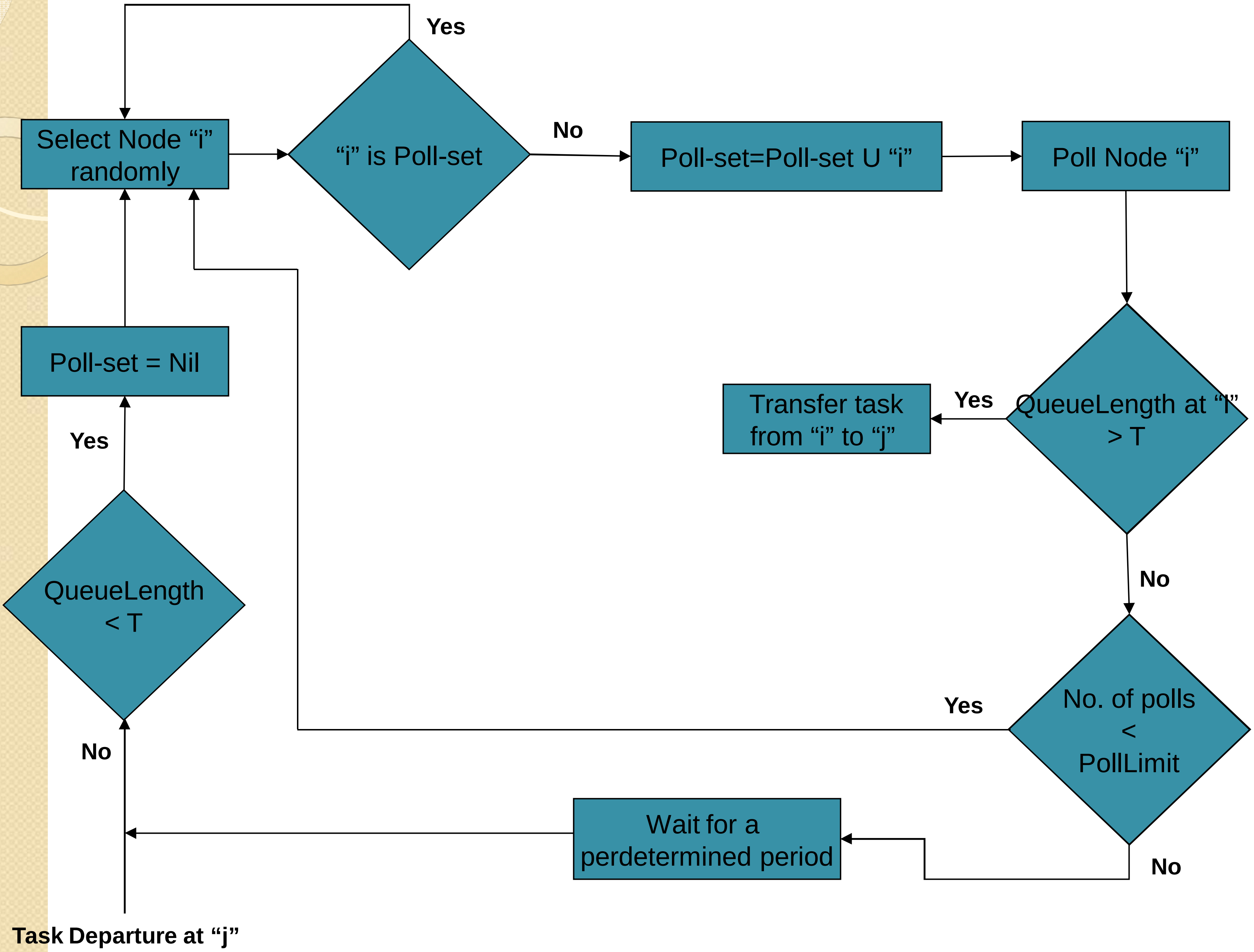
No

Queue the task locally

No

Receiver-Initiated Algorithms

- Initiated from an underloaded node (receiver) to obtain a task from an overloaded node (sender)
  - Transfer Policy
    - Triggered when a task departs
  - Selection Policy
    - Same as the previous
  - Location Policy
    - A node selected at random is polled to determine if transferring a task from it would place its queue length below the threshold level, if not, the polled node transfers a task.
  - Information Policy
    - A demand-driven type
  - Stability
    - Do not cause system instability in high system load, however, in low load it spare CPU cycles
    - Most transfers are preemptive and therefore expensive

Yes
Select Node “i” randomly
“i” is Poll-set
No
Poll-set=Poll-set U “i”
Poll Node “i”
Poll-set = Nil
Transfer task from “i” to “j”
Yes
QueueLength at “i” > T
Yes
QueueLength < T
No
Yes
No. of polls < PollLimit
No
Wait for a perdetermined period
No
Task Departure at “j”

Symmetrically Initiated Algorithms

- Both senders and receivers search for receiver and senders, respectively, for task transfer.
- The Above-Average Algorithm
  - Transfer Policy
    - Thresholds are equidistant from the node’s estimate of the average load across all node.
  - Location Policy
    - Sender-initiated component: Timeout messages TooHigh, TooLow, Accept, AwaitingTask, ChangeAverage
    - Receiver-initiated component: Timeout messages TooLow, LooHigh, Accept, AwaitingTask, ChangeAverage
  - Selection Policy
    - Similar to both the earlier algorithms
  - Information Policy
    - A demand-driven type but the acceptable range can be

Adaptive Algorithms

- A Stable Symmetrically Initiated Algorithm
  - Utilizes the information gathered during polling to classify the nodes in the system as either Sender, Receiver or OK.
  - The knowledge concerning the state of nodes is maintained by a data structure at each node, comprised of a senders list, a receivers list, and an OK list.
  - Initially, each node assumes that every other node is a receiver.
  - Transfer Policy
    - Triggers when a new task originates or when a task departs.
    - Makes use of two threshold values, i.e. Lower (LT) and Upper (UT)
  - Location Policy
    - Sender-initiated component: Polls the node at the head of receiver's list
    - Receiver-initiated component: Polling in three order
      - Head-Tail (senders list), Tail-Head (OK list), Tail-Head (receivers list)
  - Selection Policy: Newly arrived task (SI), other approached

Task Migration

- Receiver-initiated task transfers can improve system performance at high system loads.
- Receiver-initiated transfers require preemptive task transfer.
  - Task Placement refers to the transfer of a task that is yet to begin execution to a new location and start its execution there.
  - Task Migration refers to that transfer of a task that has already begun execution to a new location and continuing its

Task Migration (contd.)

- Steps involved in Task Migration
  - State Transfer
    - The transfer of the task's state including information e.g. registers, stack, ready/blocked, virtual memory address space, file descriptors, buffered messages etc. to the new machine.
    - The task is frozen at some point during the transfer so that the state does not change further.
  - Unfreeze
    - The task is installed at the new machine and is put in the ready queue so that it can continue executing.

Issues in Task Migration

- State Transfer
- Location Transparency
- Structure of a Migration Mechanism
- Performance

State Transfer

- **The Cost**
  - To support remote execution, obtaining and transferring the state, and unfreezing the task.
- **Residual Dependencies**
  - Refers to the amount of resources a former host of a preempted or migrated task continues to dedicate to service requests from the migrated task.
- **Implementations**
  - The V-System
    - Attempts to reduce the freezing time of a migrating task by precopying the state.
    - The bulk of the task state is copied to the new host
    - It increases the number of messages that are sent to new host
  - SPRITE
    - Makes use of the location-transparent file access mechanism provided by its file system
    - All the modified pages of the migrating task are swapped to file server
  - ACCENT
    - Reduction in migration is achieved by using a feature called Copy-on-Reference
    - The entire virtual memory address space is not copied to the new host

Location Transparency

- Services that are provided to user processes irrespective of the location of the processes and services.
- In distributed systems, it is essential that the location transparency by supported.
- Location transparency in principle requires that names (e.g. process names, file names) be independent of their location (i.e. host names).
- Any operation (such as signaling) or communication that was possible before the migration of a task should be possible after its migration
- Example – SPRITE – Location Transparency Mechanisms
  - A location-transparent distributed file system is provided
  - The entire state of the migrating task is made available at the new host, and therefore, any kernel calls made will be local at new host

Structure of a Migration Mechanism

- Issues involved in Migration Mechanisms
  - Decision whether to separate the policy-making modules from mechanism modules
    - It has implications for both performance and the ease of development
    - The separation of policy and mechanism modules simplifies the development efforts
  - Decision to where the policy and mechanisms should reside
    - The migration mechanism may best fit inside the kernel
    - Policy modules decide whether a task transfer should occur, this can be placed in the kernel as well
  - Interplay between the task migration mechanism and various other mechanisms
    - The mechanisms can be designed to be independent of one another so that if one mechanism’s protocol changes, the other’s need not

Performance

- Comparing the performance of task migration mechanisms implemented in different systems is a difficult task, because of the different,
  - Hardware
    - SPRITE consists of a collection of SPARCSTATION 1
    - CHARLOTTE consists of VAX/11-750 machines
  - Operating systems
  - IPC mechanism
  - File systems
  - Policy mechanisms

Scope of Research

- Energy-Aware Scheduling of Distributed Systems Using Cellular Automata
- Challenges in parallel job scheduling

Application

- place scheduling applications behind a log-in page and limit functionality depending on the user.
- eliminating redundant data entry and saving time.