MP4: Dynamic LoadBalancer
Problem Overview

- Underutilized Nodes
- Overloaded Nodes
- Heterogeneous Nodes
  - CPU
  - Memory
  - Battery
- Heterogeneous Workloads
- Transient Variations

Solution: Move Data to maintain even Utilization across Nodes
# Algorithms Classification

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Round Robin</th>
<th>Random</th>
<th>Local Queue</th>
<th>Central Queue</th>
<th>Central Manager</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overload Rejection</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fault Tolerant</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Forecasting Accuracy</td>
<td>More</td>
<td>More</td>
<td>Less</td>
<td>Less</td>
<td>More</td>
<td>More</td>
</tr>
<tr>
<td>Stability</td>
<td>Large</td>
<td>Large</td>
<td>Small</td>
<td>Small</td>
<td>Large</td>
<td>Large</td>
</tr>
<tr>
<td>Centralized/Decentralized</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>Dynamic/Static</td>
<td>S</td>
<td>S</td>
<td>Dy</td>
<td>Dy</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Cooperative</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Process Migration</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Resource Utilization</td>
<td>Less</td>
<td>Less</td>
<td>More</td>
<td>Less</td>
<td>Less</td>
<td>Less</td>
</tr>
</tbody>
</table>
Dynamic Load Balancing

- Use System State Information at runtime

Start → Update State Information → Transfer?

- Yes → Transfer Jobs → Update State Information
- No → Sleep

Flowchart diagram showing the process of dynamic load balancing.
Policies

- **Transfer Policy:** When to initiate a transfer
  - Load Index: Metric used to decide over a transfer policy (e.g. Queue Length)
- **Selection Policy:** Which jobs will be transferred
- **Location Policy:** Partner node to receive or send jobs
  - Trivial for MP4!
- **State Information Policy:** How often to update state information
  - Time: Every 30 seconds
  - Events: When CPU usage is > 80%
  - Demand Driven: When needed by one policy
Transfer Direction

- Sender Initiated
  - Overloaded Nodes
- Receiver Initiated
  - Underloaded Nodes
- Symmetrically Initiated
  - Both but higher instability
Sender Initiated Example

**Transfer Policy**

- **Start**
- **Queue Length > T**
  - Yes: **Select Node j Randomly**
  - No: **Queue Task locally**

**State Information Policy**

- **Select Node j Randomly**
  - Yes: **Poll Node j**
  - No: **Polls < Poll Limit**

**Location Policy**

- **Polls < Poll Limit**
  - Yes: **J's Queue Length < T**
  - No: **Transfer Task**

**Selection Policy:**
Consider only Newly Queued Jobs
Receiver Initiated Example

Start

Queue Length < T

Yes

Stop

No

Select Node j Randomly

Poll Node j

Yes

No

Yes

No

Polls < Poll Limit

J's Queue Length > T

Request Task

Yes

No
**MP4 Overview**

- Distributed Architecture of two nodes
  - Two Android Phones or Two Virtual Machines
    - Each VM in a separate Server
MP4 Overview

- Distributed Architecture of two nodes
  - Two Android Phones or Two Virtual Machines
- Implement a Dynamic Load Balancer
- Decide and implement each policy:
  - Transfer Policy
  - Selection Policy
  - State Information Policy
  - Location Policy Trivial!
Workload Selection and Data Decomposition

- A simple vector addition
  - Purpose of this MP is to implement the loader balancer, not to come up with complicated workload
  - Vector addition on a 1024*1024*4 double vector
- Fully Parallelizable Algorithm
  - Jobs: Chunks of equal size
  - Initially there should be 512+ jobs
System Lifecycle

- **Bootstrap**
  - Data Decomposition and Distribution

- **Processing**
  - Processing

- **Aggregation**
  - Aggregate Data
MP4 Minimal Architecture

Local Node

- Data Channel (Jobs)
  - Transfer Manager
  - State Manager

Control Channel (State)

Remote Node

- Job Queue
- Transfer Request
- Throttling
- Worker Thread

- Hardware Monitor
  - CPU State
  - Result
  - User Throttling
  - CPU Utilization
Worker Thread

Job Queue

Job

Process Job

Sleep

Finished Job

Results

Throttling

Processing (80%) Sleeping (20%)

Time

DO NOT SPIN INSTEAD OF SLEEPING!
Load Balancer

INPUT
- State Manager
- Hardware Monitor

Adaptor
- Transfer Decision: Transfer Manager
- Throttling: Worker Thread

OUTPUT
- Remote State
- Local State
Transfer Manager

● Transfer Jobs between the two phones
  ◦ Use any protocol you decide
    • TCP/IP, UDP, HTTP, etc

● Tradeoffs:
  ◦ Bandwidth
  ◦ Stability
  ◦ Performance
State Manager

- Transfer Local State to Remote phone
- Transfer Remote State to Local Phone
- Information State Policy

Tradeoffs:
- Stability
- Bandwidth
- Performance
Hardware Monitor

- Query system to monitor system state
  - CPU Usage
  - User Defined Throttling
  - Other Optional:
    - Battery State
    - Bandwidth

- Information State Policy
  - Less frequently vs More frequently
  - Event Based?
    - Android Event: Broadcast Listener
Implementation Guidelines

- Basic Architecture Required
  - Build on top of it
- Policy Design and Analysis
  - Justify, Measure and Validate
- Data Structures
  - Performance / Computational Complexity
- System Principles
  - Timers, Threads, Synchronization, Events
- Graphical User Interface
  - Runtime Parameters and Statistics
Improvements

- Not required in this MP
- Bandwidth (Transfer and State Managers)
- Information Policy Analysis
- Concurrent Worker Threads
- Graphical User Interface
- Compression