Chapter 3:-
Constructing Scalable Services

Compiled By:- Ankit Shah
Assistant Professor,
SVBIT.
Constructing Scalable Services

- Introduction
- Environment
- Resource Sharing
- Resource Sharing Enhanced Locality
- Prototype Implementation and Extension
- Conclusions and Future Study

Ankit Shah
A complex network system may be viewed as a collection of services

- Resource sharing
  - Goal: archiving maximal system performance by utilizing the available system resource efficiently
  - Propose a scalable and adaptive resource sharing service

- Coordinate concurrent access to system resources
  - Cooperation & negotiation to better support resource sharing

- Many algorithms for DS should be scalable
  - The size of DS may flexibly grow as time passes
  - The performance should also be scalable
Environment

- Complex network systems
  - Consist of a collection of WAN & LAN
  - Various nodes (static or dynamic)
  - Communication channels vary greatly by static attributes
Algorithms & techniques that work at small scale degenerate in non-obvious ways at large scale.

Many commonly used mechanisms lead to intolerable overheads or congestion when used in systems beyond a certain size.

Topology dependent scheme or an algorithm which is system-size dependent are not scalable.

**Scalability**

- System’s ability to increase speedup as the number of processors increase.

**Speedup** measures the possible benefits of a parallel performance over a sequential performance.

**Efficiency** is defined to be the speedup divided by number of processors.
Design Principles of OS for Large Scale Multicomputers

- Design a distributed system
  - Want its performance to grow linearly with the system size
  - The demand for any resource should be bound by a constant which is independent of the system size
  - DSs often contain centralized elements (like file servers)
    - Should be avoided
  - Decentralization also assures that there is no single point of failure

Ankit Shah
Isoefficiency

- The function which determines the extent at which the size of the problem can grow as the number of processors is increased to keep the performance constant
  - Disadvantage: its use of efficiency measurements and speedup
- Indication for parallel processing improvement over sequential processing, rather than means for comparing the behavior of different parallel systems

Ankit Shah
Isoefficiency and Isospeed (2)

- **Scalability**
  - An inherent property of algorithms, architectures, and their combination
  - An algorithm machine combination is scalable if the achieved average speed of the algorithm on a given machine can remain constant with increasing number of processors, provided the problem size can be increased with the system size

- **Isospeed**
  - $W$ amount of work with $N$ processors
  - $W'$ amount of work with $N'$ processors for the same average speed, for the same algorithm
  - $W' = \frac{(N' \cdot W)}{N}$
  - The ratio between amount of work & number of processors is constant
Scalability Measurement

- RT: response time of the system for a problem size W
- W: the amount of execution code to be performed measures in the number of instructions
- RT’: system response time for the problem of an increased size W’ being solved on the N’-sized system (N’>N)
- Scalability

\[ S = \begin{cases} \frac{RT'}{RT} & \text{if } \frac{RT'}{RT} < 1 \\ 1 & \text{otherwise} \end{cases} \]
Weak Consistency

- The environment complex to handle
  - High degree of multiplicity (scale)
  - Variable fault rates (reliability)
  - Resources with reduced capacity (mobility)
  - Variable interconnections resulting in different sorts of latencies

- Weak consistency
  - Allow inaccuracy as well as partiality
  - State info regarding other workstations in the system is held locally in a cache

- Adaptive resource sharing
  - Must continue to be effective & stable as the system grows
Assumptions Summary

- Full logical interconnection
- Connection maintenance is transparent to the application
  - Nodes have unique identifiers numbered sequentially
- Non negligible delays for any message exchange
Model Definition and Requirements

- **Purpose of resource sharing**
  - Achieve efficient allocation of resources to running applications
  - Map & remap the logical system to the physical system

- **Requirements**
  - Adaptability
  - Generality
  - Minimum overhead
  - Stability
  - Scalability
  - Transparency
  - Fault-tolerance
  - Heterogeneity

Ankit Shah
Resource Sharing

- Extensively studied by DS
- Load sharing algorithms provide an example of the cooperation mechanism required when using the mutual interest relation
  - Components
    - Locating a remote resource, information propagation, request acceptance, & process transfer policies
- Decision is based on weakly consistent information which may be inaccurate at times
- Adaptive algorithms adjust their behavior to the dynamic state of the system

Ankit Shah
Performance of location policies with different complexity levels on load sharing algorithms

Random selection

- Simplest
- A lot of excessive overhead is required for the remote execution attempts
Threshold policy
- Probe a limited number of nodes
- Terminate the probing as soon as it finds a node with a queue lengths shorter than the threshold
- Substantial performance improvement

Shortest policy
- Probe several nodes & then selects the one having the shortest queue, from among those having queue lengths shorter than the threshold
- No added value to looking for the best solution but rather an adequate one

Advanced algorithms may not entail a dramatic improvement in performance
Flexible Load Sharing Algorithm

- A location policy: similar to Threshold algorithm
- Using local information which is possibly replicated at multiple node
- For scalability, FLS divides a system into small subsets which may overlap
- Not attempt to produce the best possible solution, but it offers instead an adequate one at a fraction of the cost
- Can be extended to other matching problems in DSs
Algorithm Analysis (1)

□ Qualitative evaluation
  ▪ Distributed resource sharing are preferred for fault-tolerance and low overhead purposes

□ Information dissemination
  ▪ Use information of system subset

□ Decision making
  ▪ Reduce mean response time to resource access requests
Algorithm Analysis (2)

- Quantitative evaluation
  - Performance and efficiency tradeoff
    - Memory requirement for algorithm constructs
    - State dissemination cost in terms of the rate of resource sharing state messages exchanged per node
    - Run-time cost measured as the fraction of time spent running the resource access software component
    - Percent of remote resource accesses out of all resource access requests
  - Stability
    - System property measured by resource sharing hit-ratio
    - Precondition for scalability
Resource Sharing Enhanced Locality

- Extended FLS
  - No message loss
  - Non-negligible but constrained latencies for accessing any node from any other node
  - Availability of unlimited resource capacity
  - Selection of new resource providers to be included in the cache is not a costly operation and need not be constrained

Ankit Shah
State Metric

- Positive: surplus resource capacity
- Negative: resource shortage
- Neutral: not participate in resource sharing
Network-aware Resource Allocation

Node A

Node B

Node C

Node D

Node E

2 ms

7 ms

300 ms

2000 ms

Current Node

Close Node

Distant Node

Ankit Shah
Considering Proximity for Improved Performance

- Extensions to achieve enhanced locality by considering proximity

Response Time of the Original and Extended Algorithms (cache size 5)
Estimate Proximity (Latency)

- Use round-trip message
- Communication delay between two nodes
- Observation sequence period
## Estimate Performance Improvement

### Percentage of Close Allocations

<table>
<thead>
<tr>
<th>System Size</th>
<th>Original FLS(%)</th>
<th>Extended FLS(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>38</td>
<td>49</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>53</td>
</tr>
<tr>
<td>30</td>
<td>37</td>
<td>51</td>
</tr>
<tr>
<td>40</td>
<td>39</td>
<td>52</td>
</tr>
<tr>
<td>50</td>
<td>34</td>
<td>49.66</td>
</tr>
</tbody>
</table>

### Performance Improvement of Proximity Handling

<table>
<thead>
<tr>
<th>System Size</th>
<th>Even Load (%)</th>
<th>Uneven Load (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>17.99</td>
<td>12.36</td>
</tr>
<tr>
<td>20</td>
<td>21.33</td>
<td>16</td>
</tr>
<tr>
<td>30</td>
<td>19.76</td>
<td>21.67</td>
</tr>
<tr>
<td>40</td>
<td>19.15</td>
<td>21</td>
</tr>
<tr>
<td>50</td>
<td>19.45</td>
<td>18.55</td>
</tr>
</tbody>
</table>

### Scalability Metric for the Even Load Case

<table>
<thead>
<tr>
<th></th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>1</td>
<td>0.970930/96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>0.96099</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ankit Shah
Prototype Implementation and Extension

- PVM resource manager
  - Default policy is round-robin
  - Ignore the load variations among different nodes
  - Cannot distinguish between machines of different speed
- Apply FLS to PVM resource manager
Basic Benchmark on a System Composed of 5 and 9 Pentium Pro 200 Nodes (Each Node Produces 100 Processes)
Conclusions

- Enhance locality
- Factor influencing locality
  - Considering proximity
  - Reuse of state information