LBPerf: An Open Toolkit to Empirically Evaluate the Quality of Service of Middleware Load Balancing Services

Ossama Othman

Jaiganesh Balasubramanian

Dr. Douglas C. Schmidt

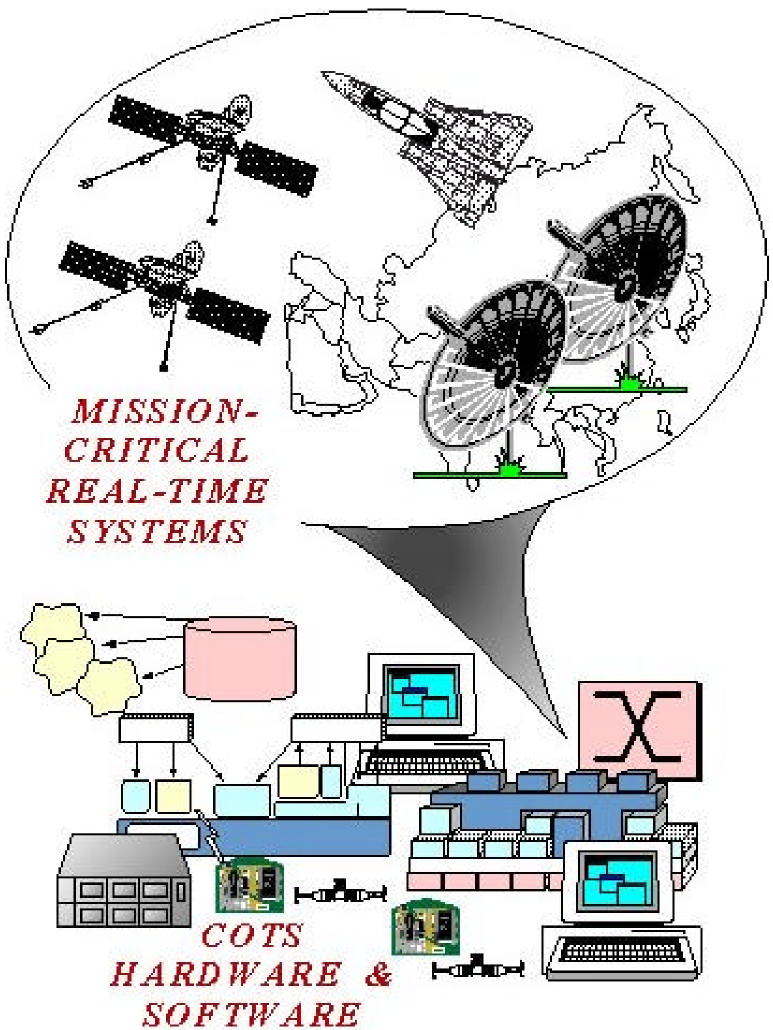
{jai, ossama, schmidt}@dre.vanderbilt.edu

Department of Electrical Engineering and Computer Science

Vanderbilt University, Nashville

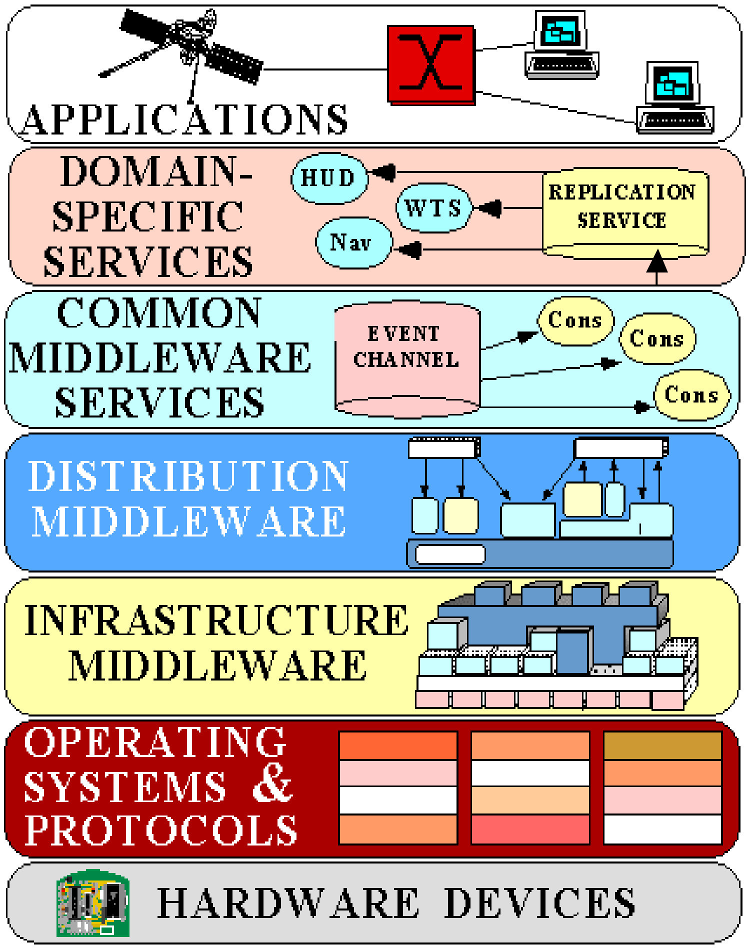
7 July 2004

Distributed Systems



- Typical issues with distributed systems
  - Heterogeneous environments
  - Concurrency
  - Large bursts of client requests
  - 24/7 availability
  - Stringent QoS requirements
- Examples of distributed systems
  - E-commerce
  - Online trading systems
  - Mission critical systems

Motivation

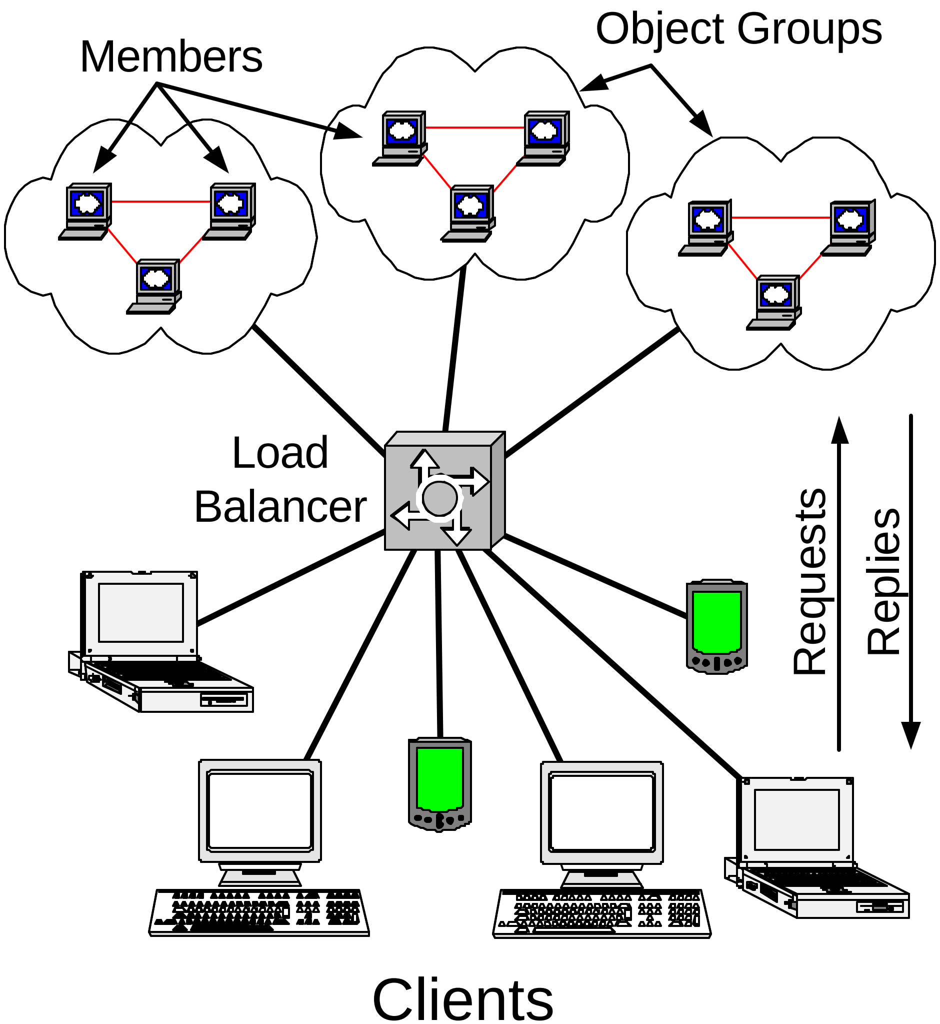


- Development and maintenance of QoS-enabled distributed systems
  - Non-trivial
  - Requires expertise that application developers often lack
- Solution: **Middleware** (*e.g.* CORBA)
  - Can shield distributed system developers from the complexities involved with developing distributed applications
  - Can facilitate manipulation of QoS requirements and management of resources

Load Balancing

- Load balancing can improve the efficiency and scalability of a distributed system
- Load balancing service can be implemented in the following layers
  - Network layer
  - OS layer
  - Middleware layer
- Why choose middleware-layer load balancing?
  - Can take into account distributed system state
  - Can take into account the system run-time behavior
  - Application level control over load balancing policies
  - Can take into account request content

Common Deployment Scenario



- **Multiple clients making request invocations**
  - Potentially non-deterministic
  - Duration called a session
- **Members**
  - Multiple instances of the same object implementation
- **Object groups**
  - Collections of members among which loads will be distributed equitably
  - Logically a single object
- **Load balancer**
  - Transparently distributes requests to members within an object group

Common Load Balancing Tasks

- Manage multiple object groups
  - Groups may be modified at run-time
- Bind clients to servers (group members)
  - Select servers based on balancing strategy configured for given object group
- Query and analyze loads
  - Pulled
    - Loads retrieved from monitoring object
  - Pushed
    - Load pushed to load balancer from monitoring object
- Rebalance loads across group members
  - Rebind clients to other servers

Inter-task Affects

- Each of the common load balancing tasks can affect the performance of the others
  - Client binding bursts can degrade load rebalancing performance
  - High frequency load reporting can degrade client binding responsiveness
  - Costly load balancing strategies can consume resources needed to respond to object group membership changes
  - Similarly for other inter-task combinations
- Execution of some tasks may starve others

Overall Performance Evaluation Considerations

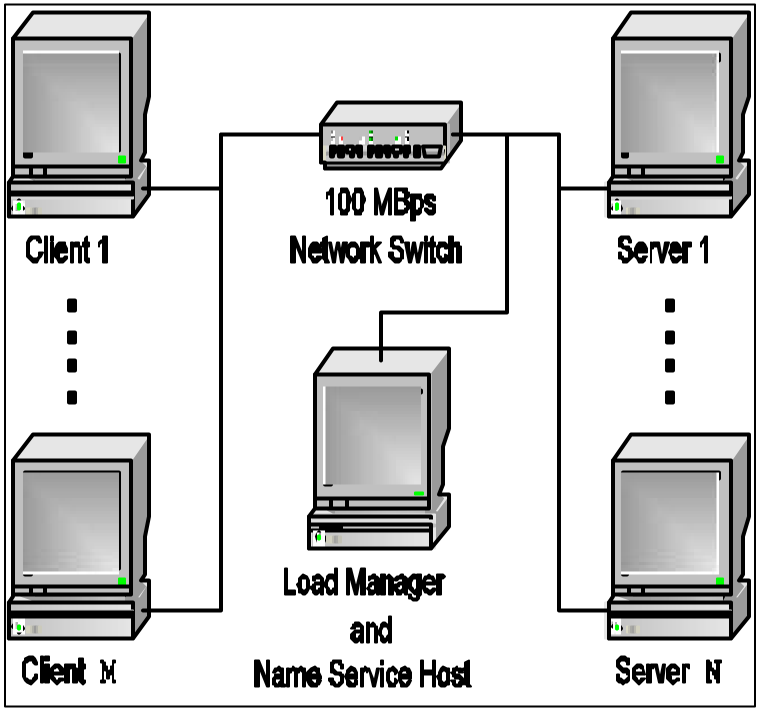
- Each of the load balancing tasks may execute non-deterministically
  - Non-determinism should be reproduced to accurately reflect true deployed run-time behavior
- Some tasks may be less critical than others
  - Give less weight to less performance critical tasks
- How do different application workloads affect load balancing performance
  - Different types of workloads incur different behavior and responsiveness from the load balancer

Evaluating Load Balancing Strategies

- Load balancing strategies can be either adaptive or non-adaptive
- Load balancing strategies can employ various run time parameters and load metrics
  - Strategies may perform differently under different parameter configurations
- Determining the appropriate load balancing strategies for different classes of distributed applications is hard without the guidance of comprehensive performance evaluation models, systematic benchmarks and empirical results

Introducing LBPerf

- LBPerf is an open source benchmarking tool suite for evaluating middleware load balancing strategies
- Supports range of adaptive and non-adaptive load balancing strategies
- Tune different configurations of middleware load balancing, including choosing different load balancing strategies and run-time parameters associated with those strategies
- Evaluate strategies using different metrics like throughput, latency, CPU utilization



Workload Characterization (1/2)

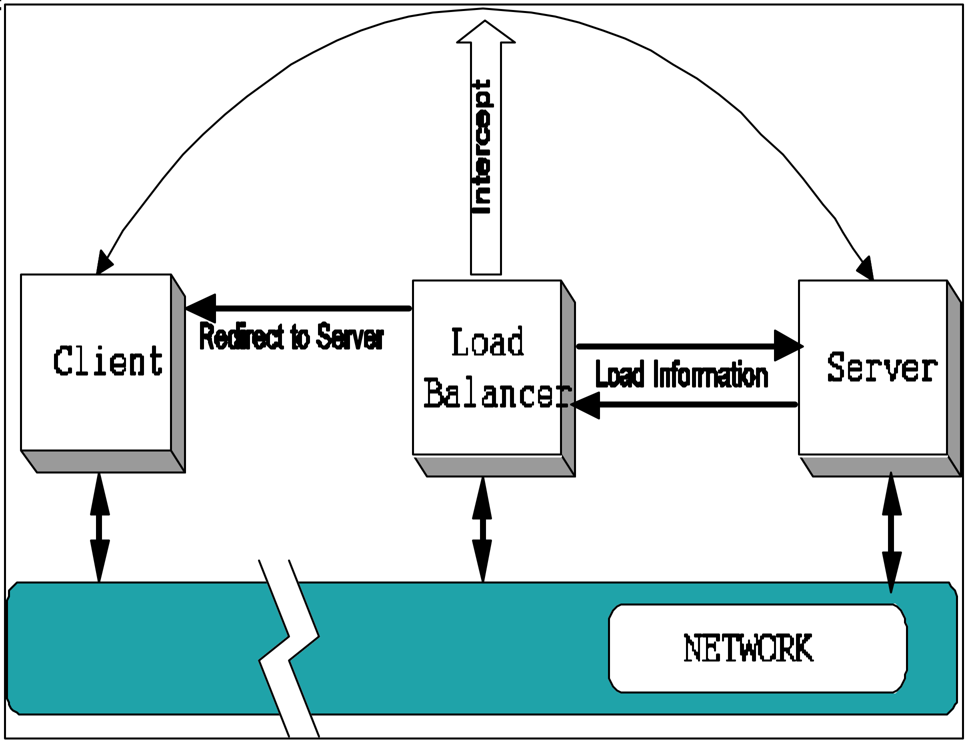
- Empirical evaluations not useful unless the workload is representative of the context in which the load balancer is deployed
- Workload model could be any of the following:
  - Closed analytical network models
  - Simulation models
  - Executable models
- Our benchmarking experiments are based on the executable workload models

Workload Characterization (2/2)

- Executable models can be classified as:
  - Resource type
    - Characterized by the type of resource being consumed
  - Service type
    - Characterized by the type of service being performed on behalf of the clients
  - Session type
    - Characterized by the type of requests initiated by a client and serviced by a server in the context of a session
- Our benchmarking experiments focused on generating session type workloads

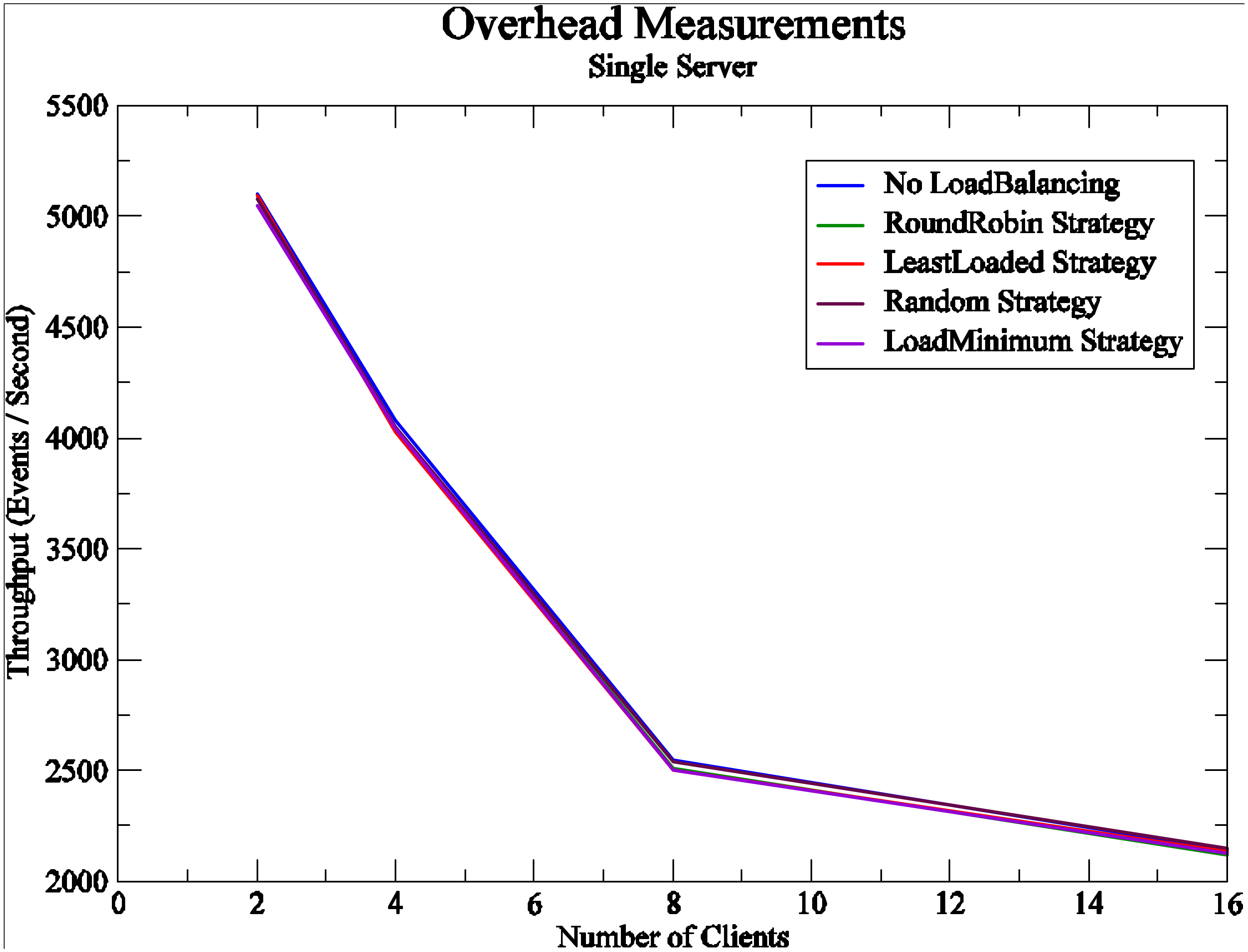
Experiment

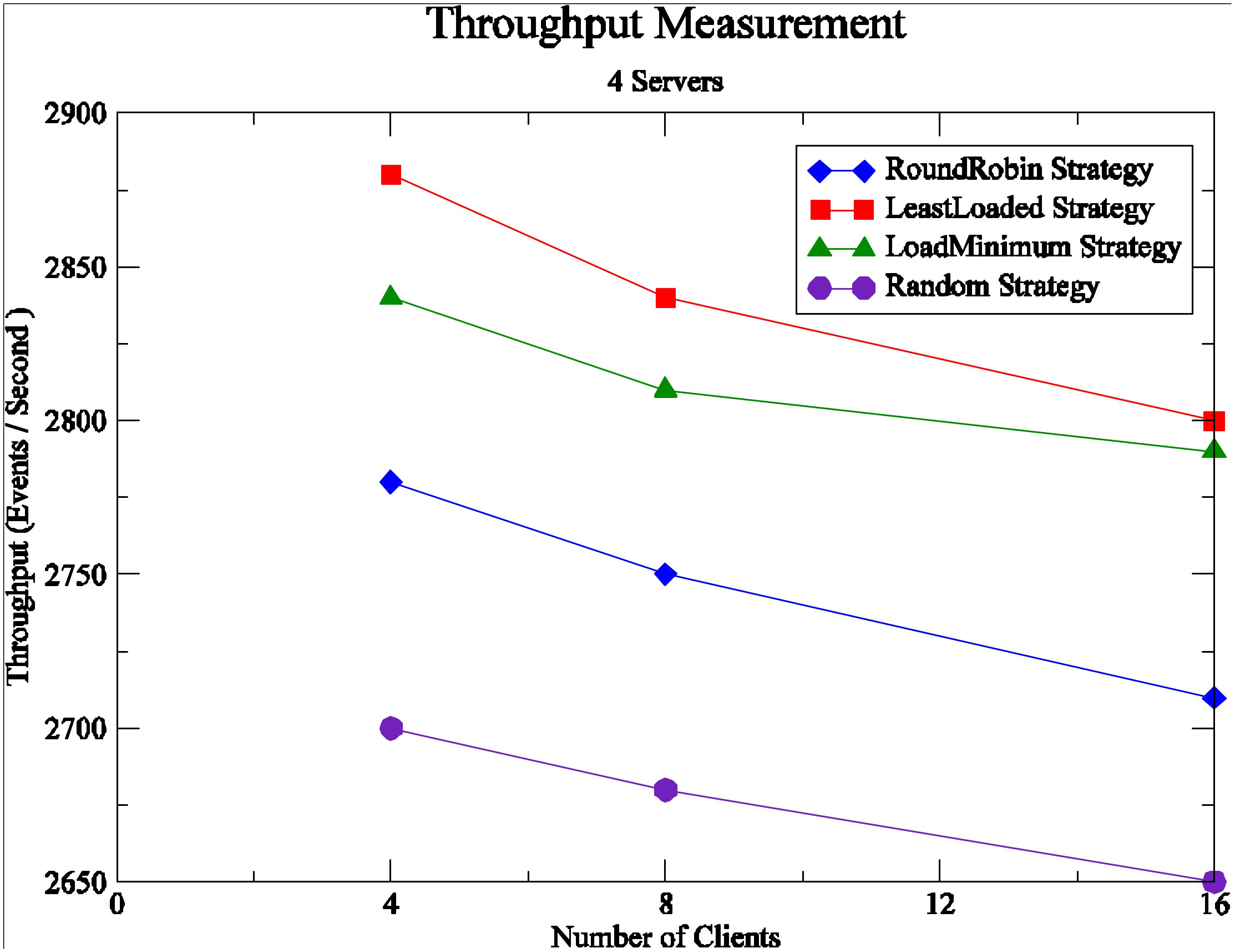
- Focus on load balancing behavior under different types of workloads
- Single threaded clients generating CPU intensive requests on the servers
- Experiments repeated for different strategies
  - Round Robin
  - Random
  - Load Minimum
  - Least Loaded
- Measurements
  - Throughput
    - The number of requests processed per second by the server
  - CPU utilization
    - CPU usage percentage

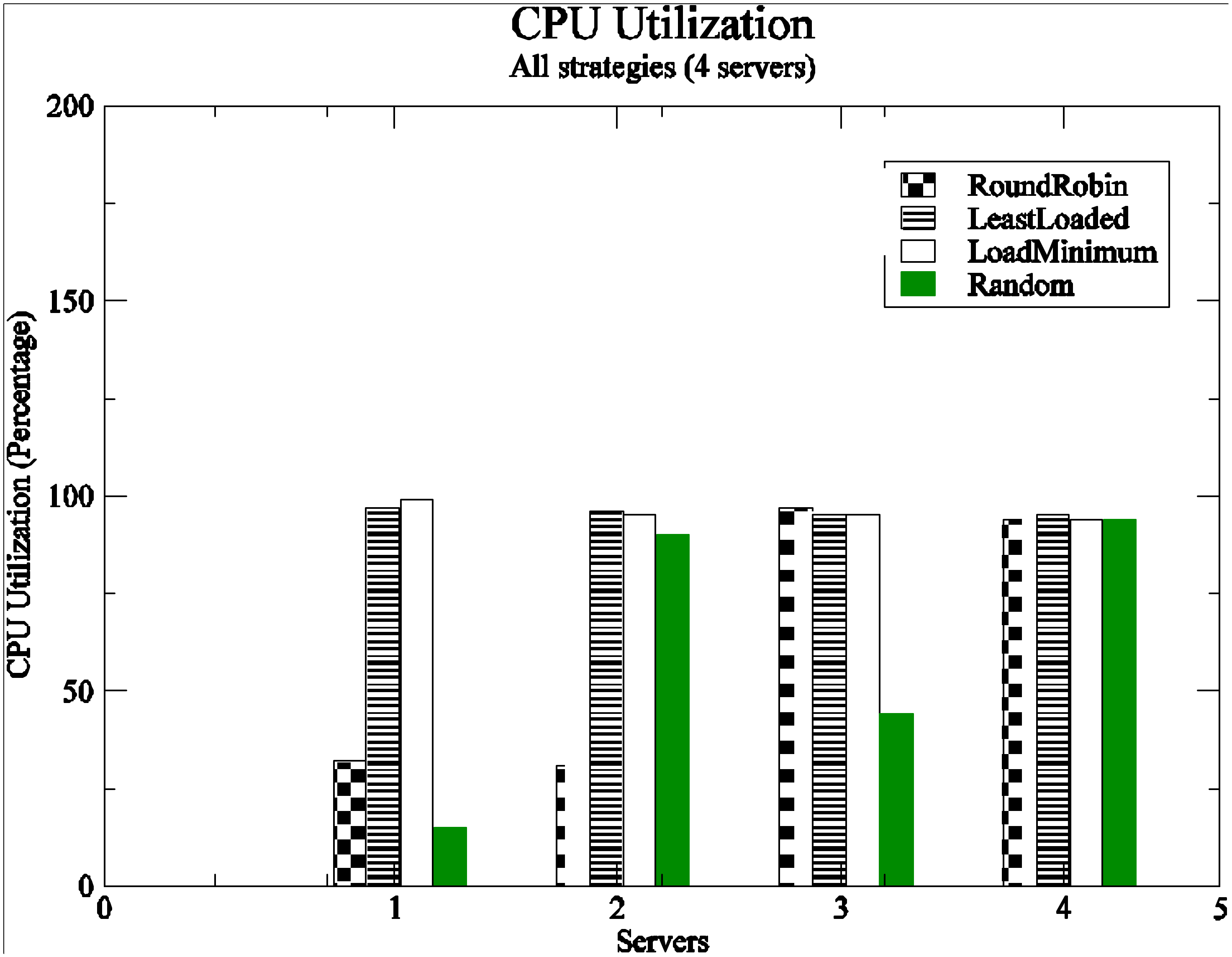


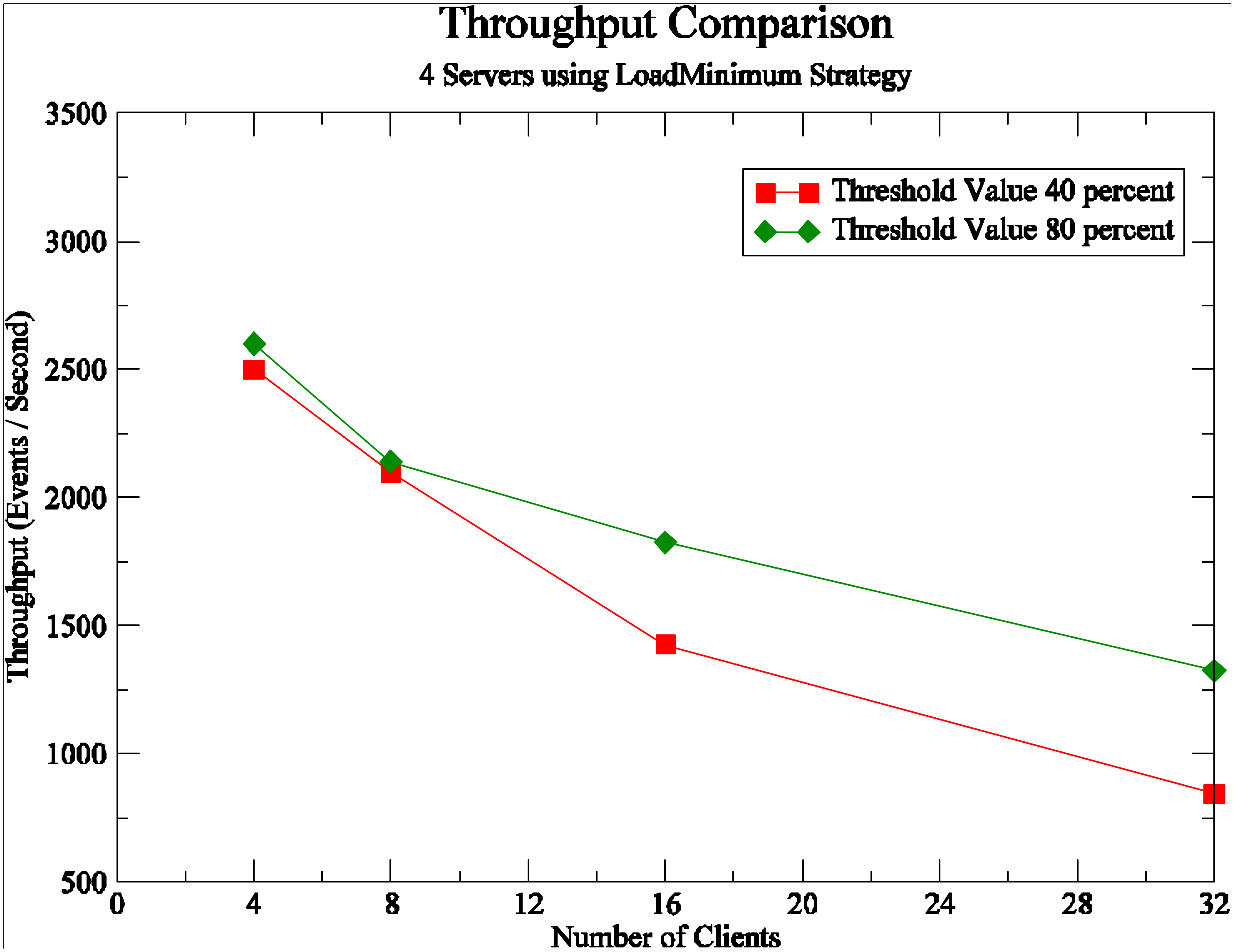
Run-time configurations

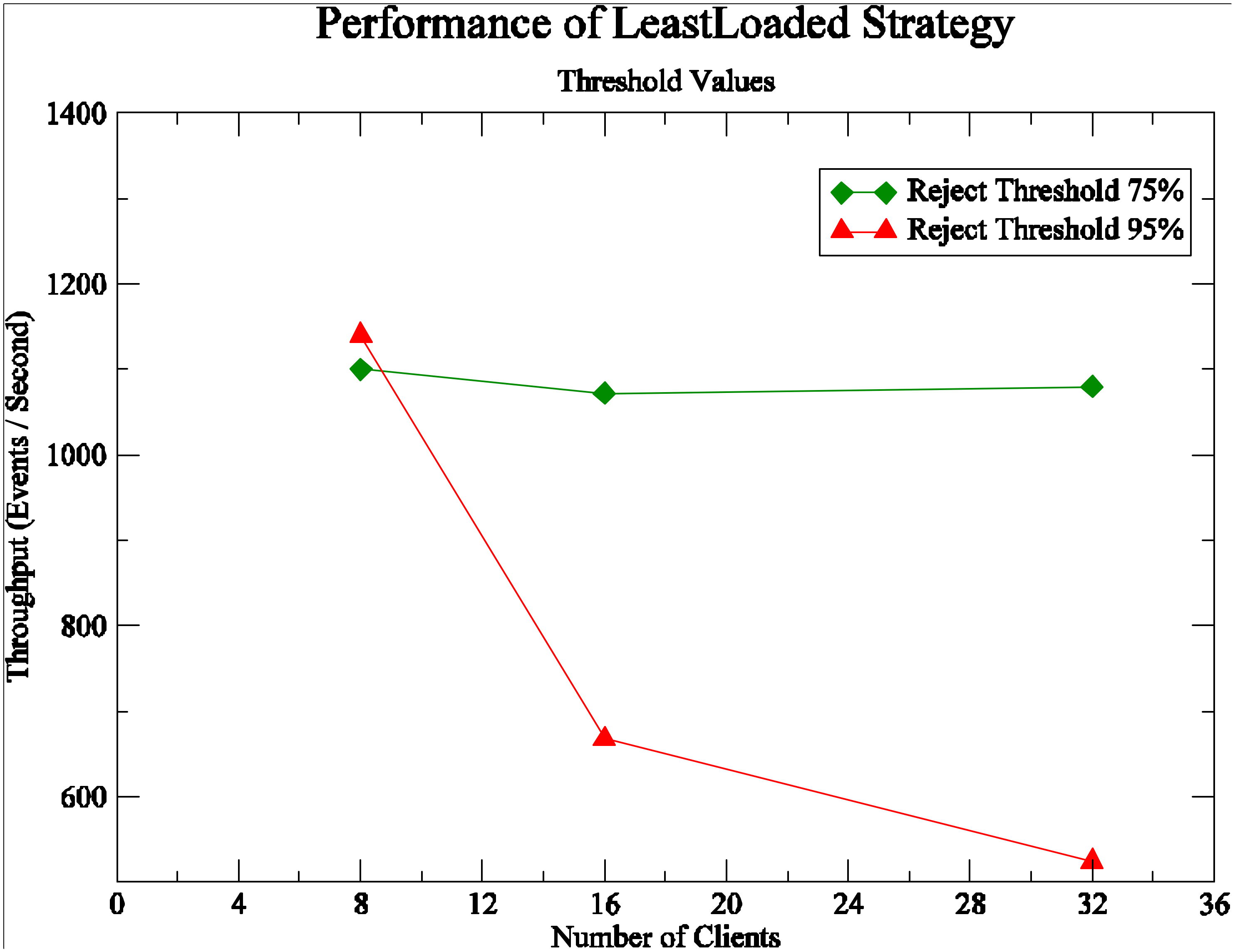
- Load reporting interval
  - The time between successive load information updates from the server location to the load balancer
- Reject threshold value of the Least Loaded strategy
  - The load at which the servers will stop receiving additional requests
- Critical threshold value of the Least Loaded strategy
  - The load at which the servers will start shedding loads
- Migration threshold value of the LoadMinimum strategy
  - The difference between the most loaded machine and the least loaded machine to trigger a migration of load from the most loaded machine to the least loaded machine
- Dampening value
  - The fraction of the newly reported load that will be considered for fresh load balancing decisions


Overhead Measurements
Single Server
Throughput (Events / Second)
5500
5000
4500
4000
3500
3000
2500
2000
0
2
4
6
8
10
12
14
16
Number of Clients
No LoadBalancing
RoundRobin Strategy
LeastLoaded Strategy
Random Strategy
LoadMinimum Strategy


Throughput Measurement
4 Servers
2900
2850
2800
2750
2700
2650
Throughput (Events / Second)
0
4
8
12
16
Number of Clients
RoundRobin Strategy
LeastLoaded Strategy
LoadMinimum Strategy
Random Strategy


CPU Utilization
All strategies (4 servers)
CPU Utilization (Percentage)
200
150
100
50
0
0
1
2
3
4
5
Servers
RoundRobin
LeastLoaded
LoadMinimum
Random


Throughput Comparison
4 Servers using LoadMinimum Strategy
Threshold Value 40 percent
Threshold Value 80 percent
Throughput (Events / Second)
3500
3000
2500
2000
1500
1000
500
0
4
8
12
16
20
24
28
32
Number of Clients


Performance of LeastLoaded Strategy
Threshold Values
1400
1200
1000
800
600
Throughput (Events / Second)
Reject Threshold 75%
Reject Threshold 95%
0
4
8
12
16
20
24
28
32
36
Number of Clients

Results Analysis

- Adaptive load balancing strategies generally perform better than non-adaptive load balancing strategies in the presence of non-uniform loads
- Least Loaded strategy is better than the other three strategies under such loading conditions
- Reducing the number of client migrations is necessary for achieving maximum performance
  - Client session migrations are not always effective
- Need to maintain system utilization to enable more predictive behavior of the strategies

Future Work

- Extend LBPerf to evaluate other types of workloads
- Use the empirical results as a learning process to understand the nuances of the different run-time behaviors of adaptive load balancing strategies
- Use observations from the learning process to train an operational phase by developing self-adaptive load balancing strategies that can dynamically tune the run-time parameters according to the load being experienced
- Implement non-deterministic traffic generator
  - Client requests
  - Load reports
  - Membership changes

Concluding Remarks

- While load balancing can improve distributed application performance significantly, determining the optimal load balancing configuration is non-trivial
  - Different load balancing strategies may behave worse than others under different types of workloads
  - Similar behavior may be observed when utilizing different load metrics
  - Improperly tuned load balancing strategy parameters can reduce effectiveness of the strategy, thus having a negative impact on performance and overall system scalability
  - Overall load balancer scalability and responsiveness may be degraded as the different tasks performed by a load balancer are executed

References

- Ossama Othman, Jaiganesh Balasubramanian, Douglas C. Schmidt

  " The Design of an Adaptive Load Balancing and Monitoring Service"

  http://www.dre.vanderbilt.edu/~ossama/papers/PDF/IWSAS_2003.pdf

- Jaiganesh Balasubramanian, Ossama Othman, Douglas C. Schmidt

  "Evaluating the performance of middleware load balancing strategies"

  http://www.dre.vanderbilt.edu/~jai/EDOC_2004.pdf

- Download Cygnus, LBPerf and TAO from

  http://deuce.doc.wustl.edu/Download.html